Universidad Autónoma del Estado de México

Facultad de Ingeniería
Maestría en Ciencias de la Ingeniería

Diseño de un algoritmo para aproximar el coloreo de una gráfica mediante conjuntos maximales independientes

TESIS

QUE PARA OBTENER EL TÍTULO DE:

Maestra en Ciencias de la Ingeniería

PRESENTA:

Angélica Guzmán Ponce

Tutor Académico:
Dr. José Raymundo Marcial Romero

Tutores Adjuntos:
M.C.C Héctor Montes Venegas
Dra. Rosa María Valdovinos Rosas

Toluca, México, Octubre 2017

Sólo podemos ver poco del futuro, pero lo suficiente para darnos cuenta de que hay mucho que hacer
-Alan Turing

La gente piensa que enfocarse significa decir sí a aquello en lo que te enfocas, pero no es así. Significa decir no a otras cientos de ideas buenas que hay.
-Steve Jobs

Hay un tiempo para cada cosa y una cosa para cada tiempo.
-Eclesiastés, 3

Reconocimientos

Esta Tesis es fruto del esfuerzo y convicción de continuar con el tema que desde licenciatura tuve el placer de conocer, las experiencias de estos dos años de formación en investigación son alicientes para saber que deseo continuar sobre el sendero de la generación de conocimiento.

Quiero agradecer en primer lugar a toda mi familia que siempre ha estado conmigo y me ha apoyado en todas mis decisiones académicas, a Elizabeth Ponce y Catalina Cuautle que me formaron como persona y siempre han estado, a quiénes siempre les estaré agradecida por todo, a mis padrinos Edith Ponce y Juan Serrano que me hacían olvidarme del estrés y por soportar mis enfados, a mi niño Fernando Hernández por intentar mostrarle que siempre se puede aspirar a más.

Parte importante de mi formación, a mi asesor el Dr. José Raymundo Marcial Romero, a quién siempre le estaré agradecida por mostrarme el camino de la investigación, quién con su ayuda me alentó a continuar y dar el salto de licenciatura a estudios de posgrado, por su paciencia y gran ejemplo.

También quiero agradecer a la Dra. Rosa María Rosas Valdovinos, quién confío en mi, por el apoyo incondicional que me brindo, por mostrarme otra perspectiva de estudio y por compartirme su conocimiento, Gracias Dra. por transmitirme la fortaleza y ayudarme a culminar está etapa.

Agradezco a todos mis amigos, a los que en este periodo formamos una amistad, por su apoyo y aliento para continuar. A ti, que al final me ayudaste a darme cuenta que siempre puedo con más de un reto intelectual.

Gracias a todos los investigadores que fueron parte de mi formación y que me guiaron cada uno en su área. A CONACYT por la beca otorgada de estudios de maestría.

El conocimiento sino se comparte, no sirve de nada.

Resumen

Dada una gráfica no dirigida $G = (V, E)$ con un conjunto de vértices V y un conjunto de aristas E, el problema del coloreo de gráficas consiste en particionar todos los vértices de V en el mínimo número K de conjuntos (colores), con la característica de que cada par de vértices en un conjunto no compartan arista. El problema de coloreo se ha estudiado desde 3 perspectivas, la primera es el problema original, encontrar el mínimo número de colores con los cuales una gráfica puede ser coloreado, la segunda trata de decidir si una gráfica es k colorable, mientras que la última, aproxima el coloreo con respecto al mejor valor reportado en la literatura (ya que puede no conocerse el mínimo). Diversas propuestas de algoritmos se han desarrollado de manera exacta o heurísticas. En esta Tesis se propone un algoritmo de tipo heurístico resultado de la combinación de un algoritmo determinista y un algoritmo evolutivo para aproximar el coloreo de gráficas con respecto al mejor valor reportado en la literatura.

El algoritmo propuesto se evalúa con un conjunto de gráficas propuestas en la literatura. Los resultados obtenidos validan la viabilidad de la propuesta con respecto a otros algoritmos reportados en la literatura.

Abstract

Given an undirected graph $G = (V, E)$ with a set of vertices V and a set of edges E, the problem of colouring graphs consists of partitioning all vertices of V in the minimum number of k sets (colours), with the characteristic that each pair of vertices in a set do not share edge. The graph colouring problem has been studied from 3 perspectives, the first one is the original problem, finding the minimum number of colours with which a graph can be coloured, the second is to decide whether a graph is k colourable, while the last, approximates the colouring with respect to the best value reported in the literature (since the minimum is unknown). To solve the graph colouring problem there are two trends, the first implies the use of exact algorithms and the second the use of heuristic algorithms. In this thesis we propose a heuristic algorithm resulting from the combination of a deterministic algorithm and an evolutionary algorithm to approximate the graph colouring with respect to the best value reported in the literature. The proposed algorithm is evaluated with a set of already reported graphs.

Índice general

Índice de figuras

Página

Índice de tablas

Página

Índice de Algoritmos

Nomenclatura

E	Conjunto de aristas.
G	Gráfica.
GA	Algoritmo Genético.
HEA	Hybrid Evolutionary Algorithm
IE^2COL	*Improved extraction and expansion method for large graph colouring.*
$MACOL$	*Memetic algorithm for graph colouring.*
V	Conjunto de vértices.

Parámetros de gráficas

$\chi(G)$	Número cromático
$\delta(G)$	Mínimo grado
$N_G(v)$	Vecinos del vértice v

Capítulo 1

Introducción

Una gráfica $G = (V, E)$ es una tupla, donde V es un conjunto de elementos denominados vértices y E es un conjunto de aristas de la forma (v, w) donde v, w pertenecen a V [9].

El problema del coloreo de gráficas consiste en asignar un color a los vértices de la gráfica, usando el mínimo número posible de colores, con la condición de que dos vértices adyacentes reciban diferente color. Al número mínimo de colores necesarios para colorear una gráfica G se le conoce como número cromático, denotado como $\chi(G)$ [17].

El coloreo de una gráfica es uno de los primeros 22 problemas NP-completos planteados por Karp [38], que continúa en investigación, con interesantes subproblemas [29, 41, 61] y aplicaciones, por ejemplo la planificación o asignación de frecuencias [6, 56], por lo que se siguen desarrollando algoritmos que disminuyan su complejidad.

Se han realizado investigaciones en torno al problema del coloreo de una gráfica [31, 3, 7, 37, 42, 63] e inclusive se ha tratado con diversas estrategias. Se ha investigado desde dos vertientes: algoritmos exactos [4, 14, 5, 13, 8] y algoritmos heurísticos [3, 16, 17, 50, 58, 60]. Para el primer caso los algoritmos desarrollados se han caracterizado por su complejidad exponencial ya sea en tiempo o espacio con respecto al número de vértices o aristas. En tanto que para métodos heurísticos, si bien no garantizan la respuesta exacta, tienden a aproximar la solución.

Por otra parte, se han desarrollado algoritmos heurísticos y metaheurísticos [63, 33] que aproximan el coloreo de una gráfica, debido a que al asignar colores se obtiene un número que no necesariamente es el mínimo, dichos algoritmos de coloreo tienen diversas estrategias de solución con una mejora continua. Para gráficas con un número mayor a 100 vértices, se ha tenido mejores resultados de aproximación al coloreo con algoritmos heurísticos, mientras que el uso de algoritmos exactos se ha realizado en gráficas con un menor número de 100 vértices [63], debido a la complejidad exponencial de obtener el coloreo.

Una estrategia de solución heurística, está basada en la obtención de conjuntos maximales independientes, abreviado como MIS, los cuales son subconjuntos de vértices, con la característica de que ningún par de elementos de este conjunto compartan una arista y no es posible agregar un vértice más. Existen algoritmos basados en este principio, por ejemplo el propuesto por Hao y Wu [63] denominado IE^2COL, que consiste en extraer conjuntos maximales independientes con la estrategia de reconsiderar aquellos conjuntos que mejoren el coloreo. Otro ejemplo es el propuesto por De Ita et al. [32], el cual a través de vértices que no corten la gráfica y de sus no vecinos formen conjuntos maximales independientes, ambas estrategias aproximan el coloreo.

Adicionalmente, existe software que permite aproximar el coloreo de una gráfica, por ejemplo JGraphT [49] librería para ser usada en JAVA de código abierto (`https://github.com/jgrapht/jgrapht`) y Sage [1] software matemático que proporciona un módulo para el problema del coloreo de gráficas, ambos aproximan el coloreo de una sola gráfica, en comparación con la implementación realizada en [31], que puede dar solución a un conjunto de gráficas que forman una gráfica general conocido como *Forest*.

En esta Tesis, se combinan las estrategias de coloreo de gráficas IE^2COL [33] con la propuesta por Guzmán et al. [32] basada en la construcción de conjuntos máximales independientes para aproximar el coloreo de una gráfica, de tal manera que la aproximación al coloreo sea igual o mejor que los resultados de JGraphT [49].

1.1. Planteamiento del problema

El problema de coloreo de gráficas ha tenido varios enfoques de estudio, dentro de los cuales podemos encontrar:

1. Dada una gráfica el objetivo es encontrar el mínimo número de colores con los cuales puede ser coloreada, mejor conocido como número cromático [14].

2. Dada una gráfica y un entero k, decidir si la gráfica es k coloreable [4], este es un problema de decisión.

3. Dada una gráfica aproximar su coloreo con respecto al mejor valor reportado en la literatura, en esta vertiente por lo regular no se conoce el número cromático.

En específico el problema que se abordará en esta Tesis es, dada una gráfica aproximar el coloreo en base a una constante k, tomada de la literatura, la cual representa la mejor aproximación obtenida hasta el momento para una gráfica,

sin embargo, si en n iteraciones se determina que la gráfica no es k coloreable, se busca encontrar el entero más cercano a k para la cual se puede determinar que es colorable.

El desarrollo de algoritmos exactos y heurísticos para solucionar el problema del coloreo ha dejado precedente la importancia de la solución, debido a que se ha llevado a la práctica en diferentes ámbitos científicos [23, 41, 61] e inclusive para resolver sudokus [43].

1.2. Justificación

El coloreo de gráficas permite describir y abordar problemas combinatorios, transformando el problema original a uno de coloreo de gráficas. La importancia de estimar el coloreo, radica en las diferentes aplicaciones o enfoques que se le puede dar a este problema: si se postulan los problemas de coloreo de mapas, asignación de horarios, asignación de frecuencias, pruebas de impresión de circuitos, asignación de rango en satélites, entre otros [12, 56, 25, 2, 18, 26, 64]. En términos de vértices, aristas y de gráficas no dirigidas, se puede dar una solución, implementando algoritmos de aproximación al coloreo de gráficas, como el planteado por De Ita et. al. [17].

Dada la complejidad del problema, desde el punto de vista de los algoritmos exactos, las propuestas desarrolladas pueden ser usadas solo para resolver problemas con un tamaño relativamente pequeño de vértices y aristas, incluso los mejores algoritmos exactos no pueden colorear de manera óptima algunas gráficas con tan sólo 124 vértices [36, 45], por lo tanto es conveniente desarrollar nuevas estrategias que disminuyan la complejidad computacional en tiempo y espacio, por ejemplo para gráficas con más de 1000 vértices [33] se considera como mejor estrategia obtener una aproximación al coloreo basadas en heurísticas y metaheurísticas.

Se ha demostrado que realizar algoritmos híbridos mejoran los tiempos de ejecución y aproximan mejor el número de colores de una gráfica [48], estos algoritmos están basados por ejemplo en algoritmos evolutivos [22, 20], en colonia de hormigas [51], los cuales obtienen resultados con mayor o menor eficiencia, pero permiten resolver la mayoría de gráficas con más de 1000 vértices.

Por lo tanto, en esta Tesis se busca desarrollar una estrategia que mejore la aproximación al coloreo de una gráfica en comparación con Guzmán et al. [32] y que iguale o mejore los resultados obtenidos por JGraphT [49].

1.3. Objetivo

Diseñar e implementar un algoritmo que dada una gráfica G y un número k se decide si es una gráfica k coloreable, caso contrario se aproxima el coloreo de la gráfica. Para este fin se establecen los siguientes objetivos particulares:

1. Analizar los diferentes tipos de creación de árboles de expansión para construir conjuntos máximales independientes.
2. Mejorar la aproximación del coloreo de una gráfica obtenido del algoritmo propuesto por Guzmán et al. [31], para gráficas con menos de 1000 vértices tomadas de la literatura.
3. Combinar dos estrategias heurísticas de solución para aproximar el coloreo de una gráfica (Hao et al. [33] y Guzmán et al. [32]) para gráficas con menos de 1000 vértices tomadas de la literatura.
4. Validar el nuevo algoritmo propuesto respecto a los resultados obtenidos por JGraphT [49] y Guzmán et al. [32].

1.4. Hipótesis

Se puede mejorar al algoritmo propuesto por Guzmán et al. [31] a partir de la obtención de conjuntos maximales independientes, con iguales o mejores resultados que JGraphT y Guzmán et al. [32] en términos del número cromático obtenido.

1.5. Estructura de la Tesis

La Tesis consiste de cinco capítulos, el primer capítulo contextualiza el problema del coloreo de gráficas, mientras que el marco teórico y estado del arte se desarrollan en el capítulo 2. La metodología a seguir se describe en el capítulo 3, en tanto que en el capítulo 4 se presentan las pruebas y validaciones realizadas al algoritmo de aproximación, por último se presentan las conclusiones y trabajo futuro en el capítulo 5.

Capítulo 2

Marco teórico y Estado del arte

2.1. Teoría de gráficas

La teoría de gráficas inicia en 1736 con los estudios del matemático Leonhard Euler, quién publicó una solución al problema de los puentes de Königsberg, el cual consistía en encontrar un camino para recorrer siete puentes que comunican a la ciudad de Königsberg, el objetivo era recorrer todos los caminos pasando una sola vez por cada uno [21]. De manera similar, el modelado en términos de vértices y conexiones denominadas aristas, para ciertos problemas como por ejemplo redes físicas, circuitos electrónicos, caminos o estructuras moleculares, han dado explicación del por qué la teoría de gráficas es importante cuando se modelan estos problemas en gráficas [30].

2.1.1. Conceptos básicos

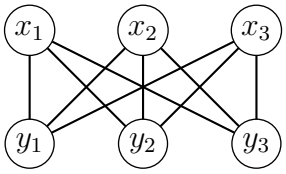
Formalmente, una *gráfica*, de acuerdo con Bondy [9], es un par ordenado $(V(G), E(G))$, donde $V(G)$ es un conjunto de elementos denominados *vértices*, $E(G)$ un conjunto de pares no ordenados de vértices denominados *aristas*.

Sea $\psi_G(e)$ una función de incidencia que asocia con cada arista de G un par no ordenado de vértices de G. Si e es una arista, y u, v son vértices tales que la función $\psi_G(e) = \{u, v\}$, entonces se dice que e es una unión de u y v, los vértices u, v se conocen como *finales* [52].

Los vértices finales de una arista se dice que son *incidentes* con la arista. Dos vértices los cuales son incidentes a una arista en común se conocen como *adyacentes*, por lo que dos vértices adyacentes se conocen como *vecinos*. Al conjunto de vecinos de un vértice v en una gráfica G se denota como $N_G(v)$ [57], por lo tanto los vértices no vecinos a un vértice v, son todos aquellos que no compartan

arista con v.

Por otra parte, una gráfica es *bipartita* si su conjunto de vértices puede ser dividido en dos subconjuntos X y Y, tal que, cada arista tiene un punto final en X y otro punto final en Y [9], se denota una gráfica bipartita G con partición (X, Y) como $G[X, Y]$. Si $G[X, Y]$ es simple y cada vértice de X está unido por un vértice en Y, entonces G es una *gráfica bipartita completa*. Un ejemplo de gráfica bipartita y que además es completa, se muestra en la Figura 2.1.



$X = [x_1, x_2, x_3]$ $Y = [y_1, y_2, y_3]$

Figura 2.1: Gráfica Bipartita.

Un *camino* es una gráfica simple cuyos vértices están organizados en una secuencia lineal, de tal manera que dos vértices son adyacentes si son consecutivos en la secuencia (Figura 2.2) [52].

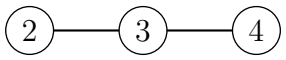


Figura 2.2: Camino.

Una arista con finales idénticos se conoce como *loop*, mientras que una arista con finales distintos se conoce como *link*, por lo que dos o más *links* con el mismo par de finales se conoce como *aristas paralelas*, es decir dos o más aristas con los mismo finales [62].

Un *Ciclo* (con al menos tres vértices), es una gráfica simple cuyos vértices pueden ser organizados en una secuencia cíclica, es decir, un vértice inicial y final se unen por una arista. La gráfica de la Figura 2.3 representa un ciclo. Un ciclo en un vértice consiste en un *loop*, mientras que un ciclo con dos vértices se conoce como arista paralela [9].

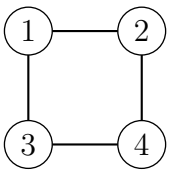


Figura 2.3: Ciclo.

Acotando el trabajo de investigación, se considerarán únicamente *gráficas simples*, es decir, aquellas gráficas que no tienen *loops* o aristas paralelas, como la que se muestra en la Figura 2.4.

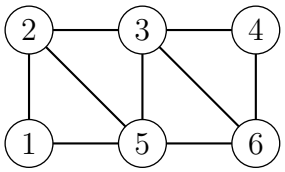


Figura 2.4: Gráfica simple.

El *grado de un vértice* v en una gráfica G, denotado por $d_G(v)$, es el número de aristas de G incidentes a v [52].

2.1.2. Operaciones con gráficas

Algunos algoritmos como el propuesto por Christofides [14] de coloreo de gráficas implementan ciertas operaciones sobre gráficas, de manera que una nueva gráfica resulte de eliminar, añadir, contraer o expandir un vértice o arista.

Dada una gráfica G y e una arista de G, se obtiene una gráfica con $m-1$ aristas, si se elimina e de G, conservando los vértices y las aristas remanentes intactas, la gráfica resultante se denota como $G \setminus e$ esta operación se conoce como *borrado de aristas* [62] y se muestra en la Figura 2.5.

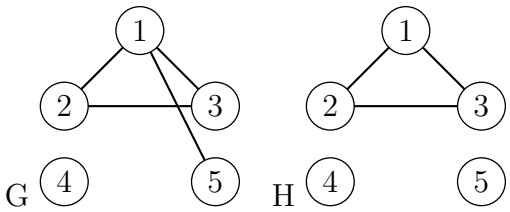


Figura 2.5: Eliminar una arista $e(1,5)$.

De manera similar sucede para el *borrado de vértices* [57], si v es un vértice de G, se obtiene una gráfica con $n-1$ vértices, de eliminar de G el vértice v junto con todas las aristas incidentes a v, la gráfica resultante se denota como $G-v$. Por ejemplo, la gráfica de la Figura 2.6, el grafo H es una gráfica formada de eliminar el vértice 1 de G.

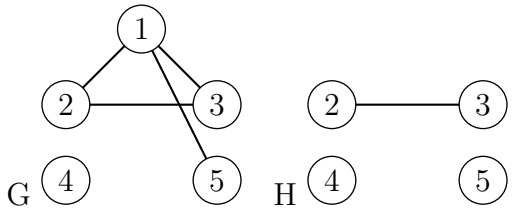


Figura 2.6: Gráfica H resultante después de eliminar el vértice 1 de G.

De manera general, una *subgráfica* de una gráfica G, es una gráfica H tal que $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$ y ψ_H es la restricción de ψ_G para $E(H)$. Se dice que G contiene a H o que H está contenida en G. En la Figura 2.7 se muestra que la gráfica H es una subgráfica de la gráfica G [9].

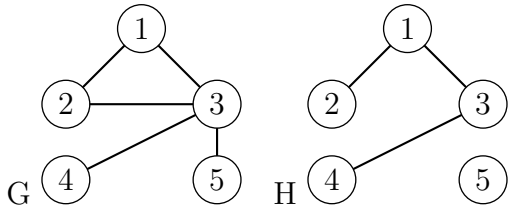


Figura 2.7: Subgráfica de una gráfica.

Una subgráfica obtenida de sólo borrar vértices, se denomina *subgráfica inducida*. Si X es el conjunto de vértices eliminados, la gráfica resultante es denotada por $G \setminus X$. Frecuentemente es el conjunto $Y := V \setminus X$ de vértices, en este caso la subgráfica es denotada por $G[Y]$ y se refiere a la subgráfica de G inducida por Y, es decir que $G[Y]$ es la subgráfica de G cuyo conjunto de vértices es Y y cuyo conjunto de aristas consiste en todas las aristas de G que tengan ambos finales en Y [52].

Otra manera de modificar una gráfica G es añadiendo un vértice v o una arista e que una dos vértices ya existentes en G, denotando estas operaciones como $G + v$ y $G + e$ respectivamente.

Existe otro tipo de operaciones denominada *contracción de aristas* [57] la cual consiste en eliminar una arista e, identificar sus extremos en sólo vértice, como se muestra en la Figura 2.8.

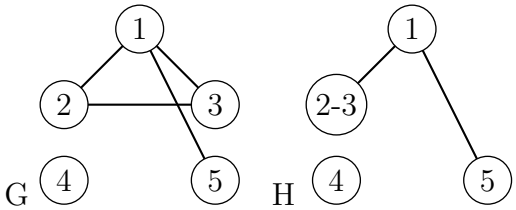


Figura 2.8: Gráfica H resultado de contraer la arista $(2, 3)$ de la gráfica G.

Por último, se puede realizar la operación de *expandir el vértice* [62] v, esto implica añadir a una gráfica G un nuevo vértice v' y unirle mediante aristas todos los vértices adyacentes a v (Figura 2.9).

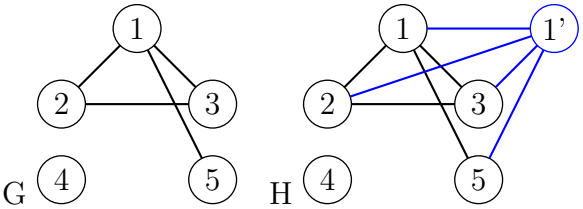


Figura 2.9: Gráfica H resulta de expander el vértice 1 en la gráfica G.

Por otro lado, un vértice de corte en una gráfica G (gráfica conectada), es un vértice v en el cual $G - v$ resulta en una gráfica desconectada, por el contrario un *vértice de no articulación*, es aquel vértice v para el que $G - v$ resulta una gráfica conectado [19].

Otro tipo de gráfica que es de utilidad en el problema del coloreo de gráficas y comúnmente usada en teoría de gráficas, es un *árbol* [57], el cual es una gráfica conexa sin ciclos, es decir, un árbol es una gráfica G, tal que para cualquiera par de vértices de G existe un único camino que los une. En la Figura 2.10 se muestra una gráfica de tipo árbol.

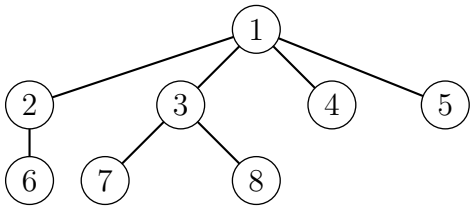


Figura 2.10: Gráfica tipo árbol.

Un árbol de expansión, contiene todos los vértices de la gráfica original sin

aristas que forman ciclos. En la Figura 2.11 se muestra un posible árbol de expansión de la gráfica de la Figura 2.4 [30].

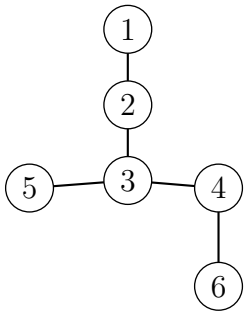


Figura 2.11: Árbol de expansión de la gráfica Figura 2.4.

El complemento $E \backslash T$ de un árbol de expansión T se conoce como *co-árbol* y se denota como $\bar{T}$, es decir, es una subgráfica de G que se forma con los vértices y aristas que forman ciclos en G y que no están en el árbol de expansión. En la Figura 2.12 se muestra el co-árbol obtenido de la gráfica de expansión de la Figura 2.11 con respecto a la gráfica de ejemplo Figura 2.4

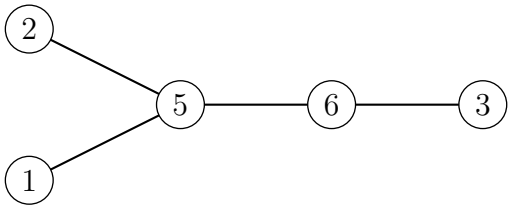


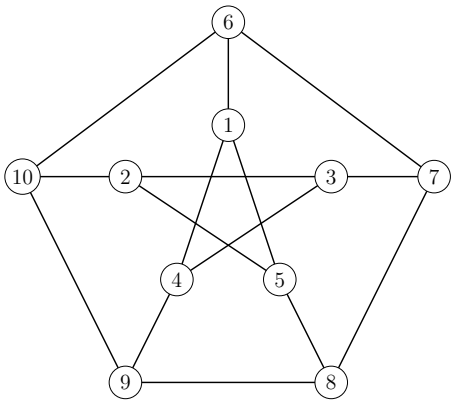
Figura 2.12: Co-árbol de la gráfica Figura 2.4.

2.1.3. Conjuntos maximales independientes

De manera general, el problema de investigación se basa en la obtención de conjuntos de vértices con una característica en específico, denominado *conjunto maximal independiente.*

Definición 2.1 *Dado $G = (V(G), E(G))$, $S \subseteq V$ es un conjunto independiente de G o también denominado conjunto estable, si para cualquier par de vértices $v_1, v_2 \in S$, $[v_1, v_2] \notin E$ [52].*

En otras palabras, S es un conjunto de vértices dónde ningún par de vértices es adyacente. Como se muestra en la Figura 2.13 los posibles conjuntos independientes que se forman de la gráfica conocida como *Gráfica de Petersen*, son S_i que se muestran a la izquierda de la figura.



$$S_1 = \{1, 2, 7, 9\}$$
$$S_2 = \{1, 3, 8, 10\}$$
$$S_3 = \{1, 2, 8\}$$
$$S_4 = \{1, 3, 9\}$$
$$S_5 = \{2, 4, 6, 8\}$$
$$S_6 = \{2, 4, 7\}$$
$$S_7 = \{3, 5, 6, 9\}$$
$$S_8 = \{3, 5, 10\}$$
$$S_9 = \{4, 5, 7, 10\}$$
$$S_{10} = \{4, 5, 6\}$$
$$S_{11} = \{1, 8, 10\}$$
$$S_{11} = \{2, 8\}$$
$$S_{12} = \{3, 6\}$$

Figura 2.13: Gráfica de Petersen con conjuntos independientes.

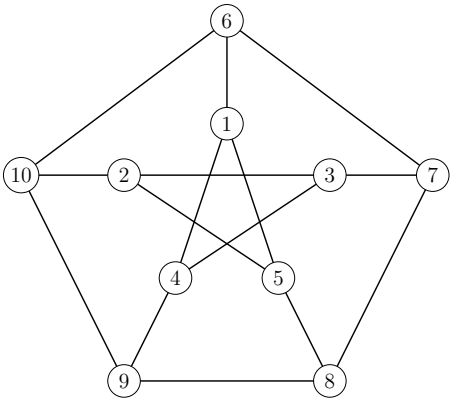
Sea $|S|$ la cardinalidad de un conjunto S y $I(G)$ el conjunto de todos los conjuntos independientes de la gráfica G.

Definición 2.2 *S es máximal abreviado como MIS, si $\forall S' \in I(G), |S| \geq |S'|$, es decir, el conjunto S no puede ser extendido a un conjunto independiente más grande [30].*

Definición 2.3 *S es máximo, si es el de tamaño más grande entre todos los conjuntos independientes en $I(G)$ [62].*

Como se muestra en la Figura 2.14 de la gráfica de Petersen los conjuntos máximos con los sombreados en gris, por otra parte, todos cumple con la propie-

dad de conjuntos máximales independientes, ya que ningún conjunto puede ser extendido sin que pierda la propiedad de ser independiente.



$S_1 = \{1, 2, 7, 9\}$
$S_2 = \{1, 3, 8, 10\}$
$S_3 = \{1, 2, 8\}$
$S_4 = \{1, 3, 9\}$
$S_5 = \{2, 4, 6, 8\}$
$S_6 = \{2, 4, 7\}$
$S_7 = \{3, 5, 6, 9\}$
$S_8 = \{3, 5, 10\}$
$S_9 = \{4, 5, 7, 10\}$
$S_{10} = \{4, 5, 6\}$

Figura 2.14: Gráfica de Petersen con MISes y conjunto máximo independiente.

2.1.4. Coloreo de gráficas

La teoría del coloreo inicia con el problema de colorear las ciudades de un mapa de tal manera que dos ciudades con un borde en común no reciban el mismo color [59]. Si se considera un mapa de la tierra (esfera) considerando sólo países, no océanos, lagos, ríos o algún otro cuerpo de agua, con la restricción de un país debe ser una continua masa en una pieza, y sin hoyos, el problema es conocer el mínimo número de colores que un cartógrafo por ejemplo necesita para poder colorear todos los mapas de este tipo, a esto se le denomino el *problema de 4-coloreo*.

En 1879, A. Kempe [39] publicó una solución al problema de 4-coloreo, dónde demostró que cualquier mapa en la esfera podría ser coloreado con cuatro colores, sin embargo P. Heawood [53], descubrió un error. La prueba de Kempe consistía en formar una secuencia de ciudades que alternaran color, justamente dos colores, denominada *"Kempe chains"*, Headwood encontró una fórmula, que estima el número cromático de cualquier superficie geométrica más complicada que una esfera.

De manera formal, el problema del coloreo de vértices de una gráfica $G = (V, E)$ está dado por un mapeo $\alpha : V \to K$, donde K es un conjunto finito de

colores $\{1, 2, 3, ..., k\}$, un coloreo de G es un par (G, α) tal que, $\alpha(V') = k$, con $k \in K$, si $v, w \in V$, $\alpha(v) \neq \alpha(w)$, si $v, w \in E$ y $V' \subseteq V$.

En otras palabras el problema del coloreo de gráficas abreviado GCP, consiste en colorear a los vértices de la gráfica, usando el mínimo número posible de colores, con la condición de que dos vértices adyacentes reciban diferente color. Un coloreo es factible, si los vértices en cada arista en E tienen asignado diferente color. Un coloreo de vértices de una gráfica G con k colores y $k \geq 1$ se denomina $k - coloreo$. Un ejemplo de una gráfica cuyos vértices están coloreados por tres colores, se muestra en la Figura 2.15.

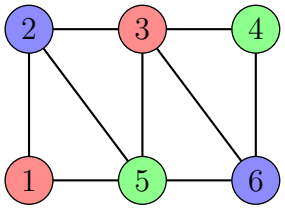


Figura 2.15: Gráfica Coloreada.

El *número cromático* de una gráfica G se define como el valor mínimo de $k \in \mathbb{N}$ tal que G es $k - coloreable$ y se denota por $\chi(G)$, si $k = \chi(G)$ se dice que la gráfica es *k-cromático*. Una gráfica completa con n vértices, es n-cromática, porque todos los vértices son adyacentes [9].

2.2. Algoritmos de búsqueda

Un Algoritmo de búsqueda está diseñado para encontrar información con ciertas propiedades en alguna estructura de datos o una posible solución de un espacio de búsqueda de un problema determinado [40].

Algoritmo voraz

Los algoritmos voraces (*greedy*), son de tipo heurístico, buscan la solución local óptima en cada ejecución, esperando encontrar una solución óptima global, es decir, se busca lo mejor para la solución de un determinado problema sin importar si la manera en cómo se obtiene sea óptimo o no [54].

Un algoritmo voraz siempre toma la solución que parece la mejor hasta el momento, por lo que no siempre se obtienen soluciones óptimas, es un método

que trabaja bien para ciertos problemas por ejemplo para el mínimo árbol de expansión, algoritmo de Dijkstra o el algoritmo de Chvátal, entre otros. De manera general las propiedades de un algoritmo voraz son [15]:

1. Determinar la estructura óptima del problema.
2. Desarrollar una solución recursiva.
3. Demostrar que si tiene una solución voraz, únicamente queda un subproblema.
4. Probar que siempre se mantiene la solución voraz.
5. Desarrollar un algoritmo recursivo que implemente una estrategia voraz.
6. Convertir el algoritmo recursivo en un algoritmo iterativo.

Depth-first search

El algoritmo de búsqueda en profundidad (*Depth-first search*) es un algoritmo para construir árboles de expansión, encontrar vértices de articulación y bloques de gráficas o para determinar si una gráfica es planar, es decir que puede ser dibujada sin que las aristas se crucen entre si [30].

Para las siguientes secciones, la gráfica que se usará para seguir los ejemplos de cómo funciona cada algoritmo se muestra en la Figura 2.16.

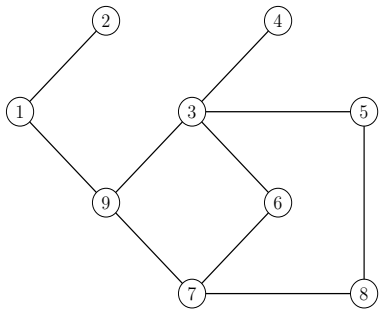


Figura 2.16: Gráfica de ejemplo.

El algoritmo de búsqueda en profundidad tiene como objetivo seleccionar aristas incidentes al vértice recientemente descubierto para lograr construir un árbol que crezca hacia abajo, si no es posible retrocede al vértice recientemente encontrado y trata de nuevo por otro camino [9].

Algoritmo 2.1 Algoritmo de búsqueda en profundidad.

Entrada: Una gráfica $G(V, E)$ conectada, un vértice de inicio $v \in V$

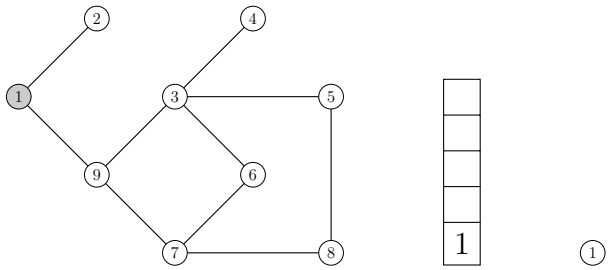
Salida: Un árbol de expansión T de G con raíz v

1: Inicializar T con el vértice v
2: Inicializar S como el conjunto de aristas incidentes a v.
3: **while** $S = \emptyset$ **do**
4: $e = dfs\_siguienteArista(G, S)$
5: w vértice final de la arista e que no pertenece a T
6: Agregar la arista e y el vértice w al árbol T
7: $actualizarFrontera(G, S)$
8: **end while**
9: **return** T

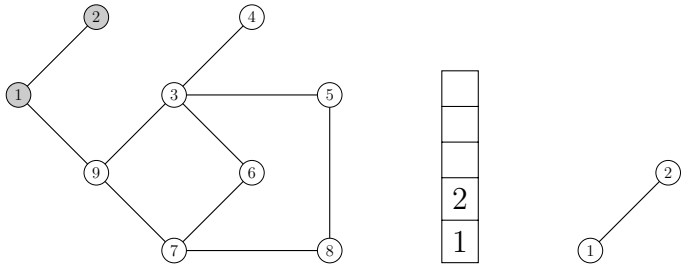
En el Algoritmo 2.1 el método $dfs\_siguienteArista$ selecciona y regresa la arista cuyo camino tiene el mayor número de vértices finales, si no hay más de una arista, entonces se selecciona una de manera prioritaria por defecto [30].

De la gráfica mostrada en la Figura 2.16, se muestra paso a paso el algoritmo de búsqueda en profundidad, generando al final un árbol de expansión T inducido de la gráfica inicial.

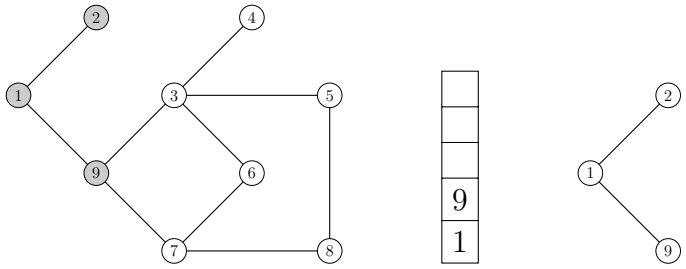
El Algoritmo 2.1 recibe un vértice de inicio denominado v, para el ejemplo, el vértice inicio es 1. Se maneja una pila con los vértices visitados, cada que se agrega un vértice a la pila este vértice se agrega al árbol T, así como la respectiva arista con el vértice que estaba en el tope de la pila, como lo muestra el siguiente ejemplo.



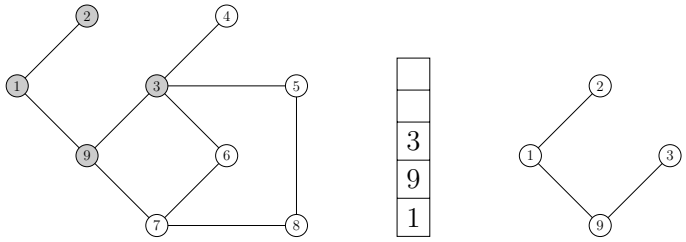
Como el vértice 1 es el primero, se introduce a la pila y se obtienen sus vecinos en la gráfica original 2 y 9, por orden numérico se introduce el vértice 2 en la pila y en el árbol T se agrega el vértice 2 con la respectiva arista con 1.



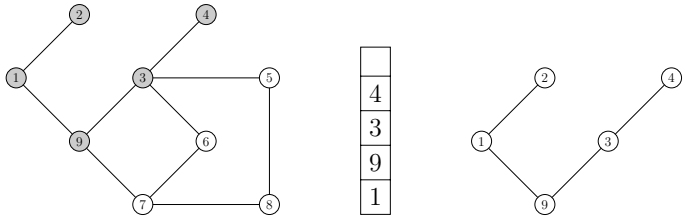
El último vértice introducido en la pila es 2, se obtienen los vecinos que no hayan sido visitados, pero como ya no cuenta con más vecinos, se saca de la pila 2. El siguiente vértice en el tope de la pila es 1, de nuevo se obtienen sus vecinos que no han sido visitados, en este caso es el vértice 9. Se introduce 9 a la pila y se agrega el vértice 9 al árbol T, así como la respectiva arista.



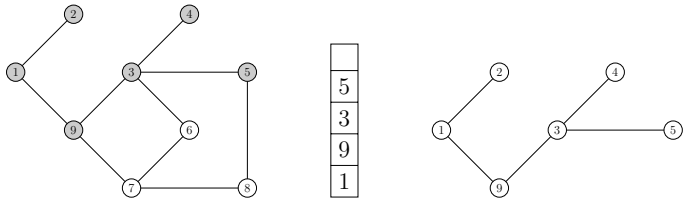
Como el vértice 9 está en el tope, se obtienen los vecinos, los cuales son los vértices 3 y 7, por orden numérico se ingresa a la pila el vértice 3.



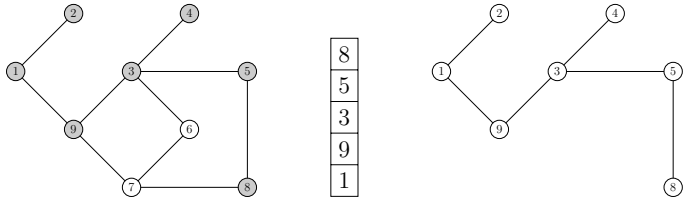
El vértice 3 está en el tope de la pila, se obtienen los vecinos de 3 que no han sido visitados, los cuales son 4, 5 y 6, por orden numérico se ingresa a la pila 4 y en el árbol T se agrega la arista $(3, 4)$



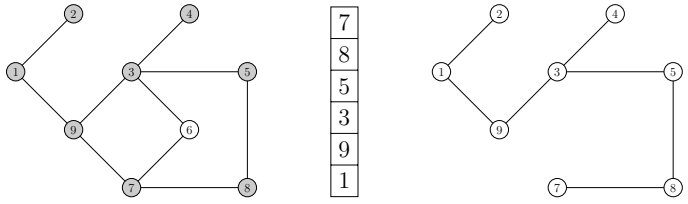
Ahora el vértice 4 se encuentra en el tope de la pila, como el vértice 3 es el único vértice adyacente a 4 pero como ya esta visitado, se saca a 4 de la pila. Quedando hasta el momento el vértice 3 al tope, de nuevo se obtienen los vértices adyacentes que no han sido visitados, estos son 5 y 6, por orden alfabético se agrega 5 a la pila y la respectiva arista $(3, 5)$ al árbol T.



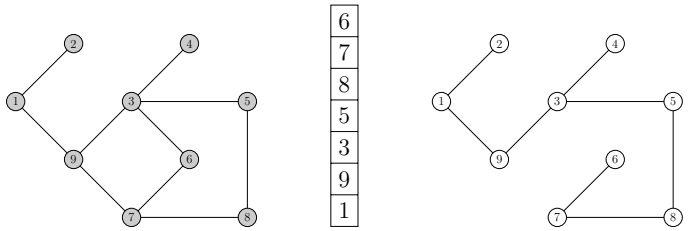
Como el vértice 5 está al tope de la pila, el vértice vecino sin haber sido visitado de 5 es 8, este se introduce a la pila y la arista $(5, 8)$ se agrega al árbol T.



El vértice que se encuentra en el tope de la pila es 8, se obtienen los vértices vecinos que no han sido visitados, para este ejemplo el vértice es 7, este se agrega a la pila y la arista $(8, 7)$ se agrega al árbol t.



Como 7 se encuentra al tope de la pila, el vértice vecino que aún no ha sido visitado es el vértice 6, por lo que se agrega a la pila y al árbol T se agrega la arista $(7, 6)$.



El vértice 6 se encuentra al tope de la pila, hasta el momento ya no se cuentan con vértices que no hayan sido visitados, por lo que se saca el vértice 6 de la pila, el siguiente vértice al tope de la pila es 7, una vez más ya no hay más vertices a visitar, se saca de la pila, así hasta vaciar la pila. Al finalizar el árbol de expansión obtenido se muestra en la Figura 2.17

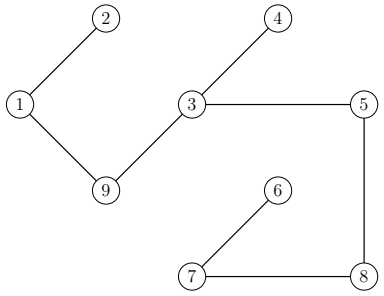


Figura 2.17: Gráfica de expansión.

Búsqueda Tabú

Uno de los algoritmo que se usa en problemas de optimización es la búsqueda Tabú, el cual es un método heurístico que explora las posibles soluciones, por medio del uso de una estructura de datos, donde una potencial solución o parte de una solución es marcada como *tabú*, es decir prohibida, el algoritmo no vuelve a visitar la posible solución en un determinado número de iteraciones [11].

La búsqueda tabú se fundamenta en resolver un problema incorporando memoria adaptativa, la cual permite implementar procesos que permitan la búsqueda de una solución económica y eficiente. Las selecciones locales son guiadas por la información obtenida durante la búsqueda. Por otro lado, sucede una exploración responsable, debido a que si en una implementación determinista o probabilística de una suposición o de una selección no adecuada se puede obtener mayor información que de una adecuada solución, entonces se obtiene una base para mejorar las estrategias de acuerdo a una búsqueda histórica [27].

2.2.1. Algoritmos Genéticos

Los algoritmos genéticos son métodos de optimización basados en la idea de Charles Darwin de acuerdo a las observaciones que presentó en su obra *El origen de las especies*, donde los organismos compiten y logran adaptarse al medio, se reproducen mejorando algunas características, tomando de manera análoga los conceptos surge el cómputo evolutivo.

Los algoritmos genéticos difieren de los métodos tradicionales de optimización en [55]:

1. Trabajan con codificaciones de los puntos del espacio de búsqueda en lugar de los puntos propiamente dichos.

2. Realizan la búsqueda a partir de una población de puntos en lugar de un sólo punto.

3. No utilizan derivadas ni otras propiedades, únicamente la propia función objetivo.

4. Se rigen mediante reglas de probabilidad, no deterministas.

5. Requieren que el conjunto de puntos en el espacio de búsqueda estén codificados mediante cadenas finitas.

De manera general en el Algoritmo 2.2 se presenta la estructura de un algoritmo genético básico, basado en el propuesto por Goldberg [28].

Algoritmo 2.2 Algoritmo Genético básico.

```
Inicio → Genera aleatoriamente una población
Adaptación → Función objetivo
while Reproducción do {Con cada nueva población}
    seleccionar → Selecciona par de cromosomas
    cruza → Producir dos descendientes
    mutación → Con cierta probabilidad se cambia algún elemento del individuo
    aceptación → Colocar los nuevos individuos en la población si cumplen con las características
end while
```

El algoritmo genético parte de una población de individuos, cada uno representa una solución factible a un problema, posteriormente se evalúa cada solución candidata de acuerdo a la función objetivo o de aptitud. Cuanto mayor sea la adaptación de un individuo al problema, mayor será la probabilidad de que sea seleccionado para reproducirse, es decir, cruza de material genético con algún otro individuo seleccionado de la misma manera, generando dos hijos, sobre los cuales se aplica un operador de mutación. El resultado será un conjunto de individuos, los cuales formarán parte de la siguiente población.

Si bien un algoritmo genético no garantiza una solución óptima del problema, al final de las iteraciones, se tiene evidencia empírica de que se aproxima la solución en un nivel aceptable [47]. Las siguientes secciones describen cada una de las heurísticas involucradas.

2.2.1.1. Codificación

Para codificar un algoritmo genético, se necesita tener una representación de las soluciones del problema, buscando una posible codificación de manera similar a la estructura de una cadena de genes, el ejemplo de codificación más empleada es por medio de números binarios (Figura 2.18), sin embargo, no es la única manera de realizarla, pues también se puede implementar números reales, letras del alfabeto, este tipo de codificación es conocida como *codificación directa* (Figura 2.19) [55].

En cualquier representación, todos los individuos tienen la misma cardinalidad (tamaño) y comparten un mismo conjunto de alelos[1], que representan en términos computacionales el problema.

El conjunto de parámetros representados por un cromosoma se denomina genotipo, éste contiene la información necesaria para la construcción de un organismo, es decir, la solución real al problema, como se muestra en la Figura 2.18.

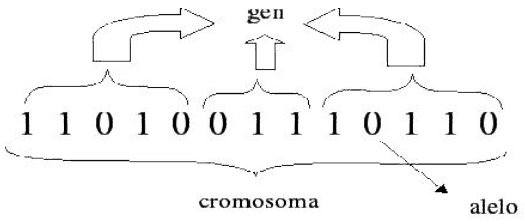


Figura 2.18: Cromosoma.

5.11	3.45	2.34	12.34	3.20

Figura 2.19: Individuo con codificación directa.

Un ejemplo se muestra en la Figura 2.19, donde los genes del individuo, representados por un número real, son parte de la solución al problema, no es necesario codificar o decodificarlo.

2.2.1.2. Función objetivo

Una vez definida la codificación, se debe establecer una función que evalúa la solución representada por un individuo, conocida como *función objetivo, función de evaluación, función de aptitud o función fitness*, la evaluación indica qué tanto

[1]Alelo es la parte minima de un individuo, en una representación binaria es un bit 0 o 1.

un individuo soluciona el problema. Para ciertos problemas, encontrar una función objetivo puede complicarse, pues dependerá del problema y el cumplimiento de ciertas características, adicionalmente la experiencia del experimentador se ve influenciada en la construcción [47]. En la mayoría de los casos se busca máximizar o mínimizar las soluciones, sin embargo no es lo único que se puede hacer, algunas otras evalúan con ciertas características.

La función objetivo suele estar diseñada para priorizar la exactitud de la solución a un problema determinado.

2.2.1.3. Operadores genéticos

Un algoritmo genético básico utiliza al menos tres operadores importantes que representan el proceso de evolución biológica: selección, cruza y mutación.

Selección

Es el proceso en el cual se seleccionan n individuos de la población de acuerdo al valor de su función objetivo para su reproducción. El principio está basado en que mientras mejor sea la aptitud del individuo, mayor es la probabilidad de ser seleccionado, de esta manera se reduce el área de búsqueda en una población descartando soluciones poco favorables [46]. Existen diferentes métodos de selección, entre los que se encuentra:

- Ruleta: Es el más simple, ya que interviene la probabilidad de que un individuo sea seleccionado por medio de su valor de la función objetivo.
- Torneo: También comúnmente usado, trata de seleccionar de manera aleatoria un determinado número de individuos de la población, compiten entre ellos y se selecciona el que tenga un mayor *fitness* (valor que se obtiene de evaluar la función objetivo) de acuerdo a las necesidades del problema.
- Selección por rango: Asigna a cada individuo un rango numérico basado en la aptitud, el proceso de selección se basa en el ranking de los individuos, a diferencia de la selección por ruleta, en la selección por rango la convergencia es lenta ya que no existe tanta diferencia entre el mejor cromosoma del resto.

Cruza

Es un proceso en el que se intercambian los componentes de los individuos seleccionados, de esta manera se producen hijos que representan nuevas soluciones. De manera similar que en genética, este operador se encarga de transferir por herencia las características de los mejores individuos [55].

El proceso más común es seleccionar un sólo punto de cruce[1] de forma aleatoria de entre el tamaño del cromosoma. Los padres son cortados en este punto y se intercambian las cuatro cadenas producidas por el punto de cruce, el cruce también se realiza con una probabilidad. El método de cruza, también se puede realizar con dos o más puntos de cruce

Una técnica más es la *cruza uniforme*, donde cada gen del descendiente se obtiene de cualquiera de los padres de manera aleatoria, con una probabilidad de selección que dependerá de un rango, si se supera se elige a un padre determinado y sino se elige al otro, esto dependerá del problema [28].

La cruza uniforme normalmente es usado para una codificación directa, esto permite que los *genes* de un individuo se combinen con otro, no se implementa en una codificación binaria porque en ésta los únicos valores son 0 o 1 [46].

Mutación

Con la mutación se producen variaciones aleatorias en un cromosoma, se puede dar en cada posición de un bit en el individuo, con probabilidad normalmente pequeña [47].

Cuando la codificación es directa, se obtiene un valor aleatorio válido en el espacio de búsqueda, se promedia con el *gen* y el resultado es el nuevo *gen*, de manera similar sucede para una mutación con reemplazo completo, el único paso que se omite es realizar el promedio, el valor obtenido aleatoriamente es el nuevo *gen* del individuo.

2.3. Estado del Arte

En el ámbito de esta Investigación se han realizado numerosos estudios en torno al problema del coloreo de una gráfica e inclusive se han tratado con diversas estrategias. Para esto, la revisión del estado del arte se ha realizado desde dos vertientes: algoritmos exactos [4, 14, 5, 13, 8] y algoritmos heurísticos [3, 16, 17, 50, 58, 60].

2.3.1. Algoritmos Exactos

Un algoritmo exacto es el que siempre encuentra la solución óptima a un problema de optimización. Los algoritmos que se han desarrollado para resolver el problema

[1]Punto de cruce: es el lugar a partir del cual se intercambian los alelos de los padres.

de coloreo de gráficas se han caracterizado por su complejidad exponencial ya sea en tiempo o espacio con respecto al número de vértices o aristas. Algunos ejemplos en cuanto a los algoritmos exactos, se encuentra el desarrollado por Christofides [14] quién propuso un algoritmo que requiere de la construcción de un árbol basado en conjuntos independientes, con un tiempo de ejecución de $n!n^{O(1)}$. Este algoritmo lista todos los conjuntos independientes óptimos que colorean una gráfica.

Por otro lado, Bodlaender y Kratsch [8] diseñaron un algoritmo para determinar el número cromático de una gráfica G basado en el mínimo de los siguientes dos términos con una complejidad tanto en tiempo como en uso de memoria de orden $O(5.283^n)$:

- El mínimo sobre todos los conjuntos máximales independientes I con $|I| \geq \alpha \cdot n$ en G de $1 + \chi(G\,[V - I])$, $0 < \alpha < 1$.
- El mínimo sobre todos los conjuntos de vértices $S \subseteq V$ con $(n - \alpha \cdot n)/2 \leq |S| \leq n/2$ de $\chi(G\,[S]) + \chi(G\,[V - S])$, $0 < \alpha < 1$.

Otra propuesta es la de Björklund y Husfeldt [5] quienes obtienen el número cromático de una gráfica en un tiempo de $O(2.3236^n)$, el cual consiste en particionar en dos conjuntos U y U' al conjunto de vértices V, para crear dos matrices A y B, indexadas por U y U' respectivamente, donde el producto AB cuenta el número de MISes en G.

Por otra parte, Byskov [13] propone un algoritmo con tiempo de ejecución $O(2.4023^n)$, basado en la idea de identificar en un arreglo todos las subgráficas que sean 3-coloreables de G, posteriormente se encuentran todos los MISes de G y se verifica que cada conjunto independiente sea 3-coloreable, en caso de ser verdadero se tiene un grafo 4-coloreable.

Finalmente, ejemplificando los algoritmos exactos Beigel y Eppstein [4], diseñaron un algoritmo para verificar si una gráfica de $|V| = n$ vértices es 3-colorable. El algoritmo que presentan es de complejidad en tiempo de orden $O(1.3289^n)$ independientemente del número de aristas en la gráfica.

2.3.2. Algoritmos Heurísticos

Los algoritmos heurísticos algunas veces obtienen la solución óptima a un problema de optimización. Dada la complejidad exponencial de los algoritmos exactos, se han desarrollado métodos heurísticos, que si bien no garantizan la respuesta exacta, tienden a aproximar la solución.

Un ejemplo de tipo voraz, es el conocido como DSATUR [10] el cual consiste en:

1. Ordenar los vértices de G de manera decreciente con respecto al grado de cada vértice.
2. Asignar el primer color al vértice de mayor grado.
3. Elegir un vértice v con el grado de saturación máximo, es decir, por el número de colores diferentes a los que v es adyacente.
    - Si existe más de un vértice, seleccionar el de mayor grado en la gráfica no coloreada.
4. Colorear el vértice seleccionado con el último coloreo posible.
5. Detener el coloreo hasta que todos los vértices estén coloreados, caso contrario regresar al paso tres.

Por otro lado, De Ita et al. [16] desarrollaron un algoritmo basado en patrones (o con una estructura en específico) en donde los ciclos básicos[1] de una gráfica determinan la propiedad de una gráfica 3-colorable. El número de cláusulas, dependerá del número de ciclos básicos en la gráfica, así el algoritmo determinará si una gráfica es 3-coloreable en tiempo polinomial de acuerdo al número de ciclos básicos de una gráfica dada.

Vlasie, R. [58] desarrolló un procedimiento para verificar que se trate de gráficas 3-coloreables, el procedimiento favorece a las gráfica con conectividad y libres de 4 cliques (clique en una gráfica G no dirigida es un subconjunto de vértices de G, donde cada vértice está conectado a otro vértice del conjunto).

Por otra parte Almara'beh et al. [3] proponen tres algoritmos heurísticos para aproximar el coloreo de una gráfica, todos ellos basados en la construcción de un MIS. El primero consiste en la selección de un vértice mezclando el grado mínimo y máximo (Min_Max), el siguiente algoritmo selecciona el nodo con grado mínimo (SNMD) y finalmente el último algoritmo selecciona el grado máximo (SNXD). El resultado que ellos obtienen, indica que los algoritmos Min_Max y SNXD son mejores con base al tiempo de procesamiento y aproximación al coloreo obtenido que el algoritmo SNMD.

Algunos algoritmos heurísticos surgen de la modificación de otros, por ejemplo Al-Omari, et al. [50] modificaron dos algoritmos, donde la clave de los algoritmos se encuentra en el criterio de selección del vértice para ser coloreado. El primer algoritmo denominado *"Largest degree ordering"* (LDO), selecciona el vértice con el mayor número de vecinos, construyendo iterativamente el coloreo; el segundo algoritmo es *"Saturation degree ordering"* (SDO), donde la saturación del grado

[1] Es un camino cerrado donde no se repite ningún vértice excepto el principio y fin del camino.

de un vértice, se define como el número de vértices adyacentes de diferentes colores y por último, *"Incidence degree ordering"* (IDO), es un algoritmo donde el grado de incidencia de un vértice se define como el número de vértices adyacentes coloreados. La primera modificación combinó LDO con IDO, resultando mejor la combinación, que sólo implementar LDO. Mientras que la segunda propuesta combina los algoritmo SDO y LDO, esta combinación dio mejor resultado que si sólo se implementá SDO, además de que éste último usa el menor número de colores para el coloreo de una gráfica.

De Ita et al. [31] proponen un algoritmo de aproximación al coloreo en donde se descartan en un principio las componentes bipartitas, ya que pueden ser coloreadas al final del proceso con dos colores, además está basado en el proceso de construcción y eliminación de MISes. El algoritmo iterativamente construye y elimina un MIS, hasta que se obtiene una subgráfica 3-coloreable. Una mejora al algoritmo se presenta en [32], consiste en una vez obtenido un vértice de no articulación x, los no vecinos son ordenados descendentemente considerando el grado del vértice, todos estos vértices son candidatos a formar parte del MIS, siempre y cuando no corten la gráfica.

Algunos algoritmos heurísticos para el coloreo de gráficas han considerado instancias de gráficas con más de 1000 vértices, considerados como *gráficas grandes*, por ejemplo, Qinghua Wu. et. al. [63] propone un algoritmo denominado EXTRACOL, el cual pre-procesa la gráfica original, obteniendo de él conjuntos independientes del mismo tamaño, hasta tener una gráfica residual, al cual se le aplica el algoritmo MACOL [44] que implementa un procedimiento de búsqueda tabú con un algoritmo evolutivo. Una mejora a EXTRACOL se presenta en [33], denominado IE^2COL, el cual emplea una estrategia de extracción de conjuntos máximales independientes en lugar de remover un sólo conjunto independiente, además si un conjunto independiente es extraído equivocadamente, es decir que el conjunto no forme parte de una solución óptima de coloreo, se aplica la fase de retroceso, la cual verifica que el conjunto independiente sea óptimo, de no serlo regresa los vértices a la gráfica.

De manera precisa, un objetivo de esta investigación es combinar dos estrategias heurísticas de solución al coloreo para obtener una sola, una primera estrategia es con el uso de algoritmos genéticos mientras que la segunda estrategia pertenece a un algoritmo determinista, es decir, se sigue una serie de pasos definidos para obtener siempre que se ejecuta una misma solución.

2.3.2.1. Algoritmos Heurísticos implementados

El algoritmo de aproximación propuesto en esta Tesis se basa en el algoritmo denominado $HEAD$ [24], el cual ha sido tomado como base para algunos otros

algoritmos de aproximación al coloreo de gráficas. Por otra parte, se considera conjuntar un algoritmo determinista de igual manera aproxima una solución. En las siguientes secciones se describe los algoritmos de aproximación implementados.

Algoritmo HEA

El algoritmo propuesto por Galinier [24] implementa un algoritmo genético, conocido como *HEA (Hybrid Colouring Algorithm)* que tiene los primeros y mejores resultados de coloreo para gráficas. Para la propuesta de ésta investigación, se retoma el algoritmo para realizar el coloreo de la gráfica, el Algoritmo 2.3 describe el algoritmo de coloreo, el cual requiere un tamaño constante de la población, es decir $|P| = 20$ [24].

Algoritmo 2.3 HEA()

Entrada: Una gráfica $G = (V, E)$, un entero k

Salida: La mejor configuración s encontrada

1: P=InicializarPoblacion($|P|$)
2: **while** No condicionParo() **do**
3: (s_1, s_2)=SeleccionarPadres(P)
4: s=GPX(s_1, s_2)
5: s=LocalSearch(s, L)
6: P=ActualizarPoblacion(P)
7: **end while**

El Algoritmo 2.3 recibe una gráfica y un entero denominado k el cual indica el número de colores con los cuales se colorea una gráfica, es importante mencionar que este número dependerá de la gráfica y será tomado de los mejores resultados registrados (`http://mat.gsia.cmu.edu/COLOR04/`). Lo primero que realiza el algoritmo es construir una población inicial denominada P, posteriormente se realiza una serie de ciclos donde se seleccionan dos individuos aleatoriamente de la población denominados s_1 y s_2 (linea 3), posteriormente se realiza el método de cruza GPX (Algoritmo 2.5) de ambos individuos, éste método genera un nuevo individuo s para realizar la búsqueda Tabú (Algoritmo 2.4) por L iteraciones. El objetivo de la búsqueda Tabú es reducir el número de conflictos del individuo s, una vez obtenida una solución s con menos conflictos se actualiza la población. Para la actualización se reemplaza el peor de los dos individuos seleccionados s_1 o s_2, es decir, se elimina de la población el que tenga mayor número de conflictos y se agrega a la población el individuo s. La condición de paro es hasta un número determinado de iteraciones [24].

El proceso que inicializa la población genera $|P|$ individuos, iniciando con k clases de color vacías $V_1 = \cdots = V_k = \emptyset$, en cada paso se selecciona un vértice

$v \in V$ para el cual tenga el mínimo número de clases permitidas, es decir aquella clase para la cual no contiene ningún vértice adyacente de v), agregando v a la clase de color, este proceso no siempre asegura asignar todos los vértices a una clase, por lo que al final se asignan a clases aleatoriamente, el individuo generado es enviado a la búsqueda Tabú para reducir conflictos.

La función objetivo o función que evalúa las posibles soluciones está dada por un $k - coloreo$, $S = \{V_1, ..., V_k\}$ la función de evaluación f cuenta el número de conflictos inducidos por S tal que:

$$f(S) = \sum_{\{u,v\} \in E} \delta_{uv} \tag{2.1}$$

Donde:

$$\delta_{uv} = \begin{cases} 1, & \text{Si } u \in V_i, v \in V_j \text{ y } i = j \\ 0, & \text{De otra manera} \end{cases} \tag{2.2}$$

Un coloreo evaluado en 0 es un $k - coloreo$ legal.

Búsqueda Tabú aplicado coloreo de gráficas

Para el problema del coloreo de gráficas el algoritmo de búsqueda tabú se ha desarrollado por Hertz y de Werra [34], quienes fueron los primeros en proponerlo, sin embargo durante estos años se han desarrollado mejoras a dicho algoritmo, por ejemplo el desarrollado por Galinier and Hao [24], el cual se muestra en el Algoritmo 2.4.

Algoritmo 2.4 Búsqueda Tabú para el coloreo de gráficas.

Entrada: Una gráfica $G = (V, E)$ y una solución S_0

Salida: S_{best}

1: $S_{best} \leftarrow S_0$
2: $TabuList \leftarrow \emptyset$
3: **while** No se cumpla la condición de paro **do**
4: Seleccionar el mejor movimiento autorizado $< v, i >$.
5: Agregar a la lista Tabú la pareja $< v, s(v) >$ por tl iteraciones.
6: Realizar el movimiento $< v, i >$ en s.
7: **end while**
8: **return** S_{best}

El algoritmo 2.4 se realizará en un número L de iteraciones (condición de paro) o hasta encontrar una solución S con cero conflictos, la selección del mejor movimiento será un vértice v que se encuentre en conflicto con algún otro en su

clase original, se moverá a alguna otra clase permitida, es decir que no se encuentre en la lista Tabú, con la condición de que sea una clase en la que menos conflictos tenga el vértice v. Por lo que de la clase donde estaba v se vuelve tabú durante tl iteraciones, este valor se calcula como $tl = Random(0, \cdots, 9) + \alpha \times nb_{CFL}$, donde nb_{CFL} es el número de conflictos en la solución y α es un valor propuesto en [24] equivalente a 0.6.

Para la gráfica de la Figura 2.16 el realizar una búsqueda tabú para la solución $S = \{\{1, 2, 3\}, \{9, 7, 4, 8\}, \{5, 6\}\}$ con 3 conflictos, la lista tabú en un principio se encuentra vacía, en una primera iteración los vértices conflicto son $[8, 1, 2, 7, 9]$, seleccionando un vértice aleatorio de los conflicto, por ejemplo el vértice 2 el cual se encuentra en la clase c_0, se busca en las clases c_1, c_2 en el cual tiene menos conflictos, en este caso se elige mover el vértice 2 a la clase c_1 se calcula $tl = 10$, se realiza el movimiento y se agrega a la lista tabú el vértice 2 y la clase c_0 por 10 iteraciones, una vez realizado el movimiento, se verifica que la solución no tenga cero conflictos, se actualiza la lista tabú, reduciendo el valor tl.

Hasta el momento la solución es $S = \{\{1, 3\}, \{9, 7, 4, 8, 2\}, \{5, 6\}\}$ y la lista tabú es {2: [[0], [9.0]]}, en la siguiente iteración los vértices conflicto son $[8, 9, 7]$, seleccionando uno aleatoriamente por ejemplo el vértice 9 el cual pertenece a la clase c_1, este vértice no se encuentra en la lista tabú por lo que agregaría con su respectivo tl iteraciones quedando como {9: [[1], [7.0]], 2: [[0], [9.0]]}, el vértice 9 se moverá a la clase con la que menos conflictos tenga, en este caso a la clase c_2, se verifica que la solución no tenga cero conflictos y se actualiza la lista tabú.

Con la solución $S = \{\{1, 3\}, \{7, 4, 8, 2\}, \{5, 6, 9\}\}$, la lista tabú hasta el momento es {9: [[1], [6.0]], 2: [[0], [8.0]]}, los vértices conflicto son $[8, 7]$, seleccionando uno por ejemplo el 8 que pertenece a la clase c_1, se busca en la lista tabú, como no se encuentra se agrega {8:[[1],[3.0]],9: [[1], [6.0]], 2: [[0], [8.0]]} se mueve a la clase con menos conflictos, es decir la clase c_0, se realiza el movimiento, hasta el momento esta solución tiene cero conflictos, por lo que se finaliza el proceso y se regresa la solución $\{\{1, 3, 8\}, \{7, 4, 2\}, \{5, 6, 9\}\}$

Algoritmo GPX

El algoritmo de cruza *GPX* se realiza para dos individuos, con la siguiente configuración $s = \{V_{01}, \cdots, V_{0k}\}$, el Algoritmo 2.5 detalla el proceso de cruza de dos individuos.

Por ejemplo para dos individuos se realiza el proceso de cruza de la siguiente manera:

$$\begin{array}{l} s_1 = \{V_{11} : \{1, 2, 3\}, V_{12} : \{4, 5, 6, 7\}, V_{13} : \{8, 9, 10\}\} \\ s_2 = \{V_{21} : \{3, 4, 5, 7\}, V_{22} : \{1, 6, 9\}, V_{23} : \{2, 8, 10\}\} \\ \hline s = \{\{\}, \{\}, \{\}\} \end{array}$$

Algoritmo 2.5 GPX()

Entrada: Dos individuos, $s_1 = \{V_{11}, \cdots, V_{1k}\}$, $s_2 = \{V_{21}, \cdots, V_{2k}\}$

Salida: Un nuevo individuo $s = \{V_1, \cdots, V_k\}$

1: $s = \emptyset$
2: **for** $l(1 \leq l \leq k)$ **do**
3: **if** *lpar* **then**
4: Seleccionar el conjunto V_{1i} de mayor cardinalidad el individuo s_1
5: $s \cup V_{1i}$
6: Remover todos los vértices de V_{1i} en s_2
7: **else**
8: Seleccionar el conjunto V_{2i} de mayor cardinalidad el individuo s_2
9: $s \cup V_{2i}$
10: Remover todos los vértices de V_{2i} en s_1
11: **end if**
12: **end for**
13: Asignar aleatoriamente los vértices de $V \setminus (V_1 \cup \cdots \cup V_k)$
14: **return** s

Se toma del individuo s_1 el conjunto de mayor cardinalidad, en este caso es el conjunto $V_{12} : \{4, 5, 6, 7\}$,se agrega a la solución s, así como también se eliminan los vértices participantes en este conjunto de la solución s_2.

$$\begin{array}{l} s_1 = \{V_{11} : \{1, 2, 3\}, V_{12} : \{\}, V_{13} : \{8, 9, 10\}\} \\ s_2 = \{V_{21} : \{3\}, V_{22} : \{1, 9\}, V_{23} : \{2, 8, 10\}\} \\ \hline s = \{\{4, 5, 6, 7\}, \{\}, \{\}\} \end{array}$$

Para el siguiente paso, se considera al individuo s_2 y el conjunto de mayor cardinalidad, en este ejemplo es $V_{23} : \{2, 8, 10\}$, se agrega a la solución s y se eliminar los vértices de la solución s_1.

$$\begin{array}{l} s_1 = \{V_{11} : \{1, 3\}, V_{12} : \{\}, V_{13} : \{9\}\} \\ s_2 = \{V_{21} : \{3\}, V_{22} : \{1, 9\}, V_{23} : \{\}\} \\ \hline s = \{\{4, 5, 6, 7\}, \{2, 8, 10\}, \{\}\} \end{array}$$

Considerando al individuo s_1, el conjunto de mayor cardinalidad es $V_{11} : \{1, 3\}$, se agrega a s y se eliminan los vértices participantes de s_2.

$$\begin{array}{l} s_1 = \{V_{11} : \{\}, V_{12} : \{\}, V_{13} : \{9\}\} \\ s_2 = \{V_{21} : \{\}, V_{22} : \{9\}, V_{23} : \{\}\} \\ \hline s = \{\{4, 5, 6, 7\}, \{2, 8, 10\}, \{1, 3\}\} \end{array}$$

La solución s ya contiene las k clases de color, sin embargo aun quedan vértices sin asignar, para este ejemplo el único vértice sin asignar es 9, por lo que se agrega aleatoriamente a cualquier clase, quedando por ejemplo: $s = \{\{4, 5, 6, 7\}, \{2, 8, 10\}, \{1, 3, 9\}\}$.

Un algoritmo determinista para el coloreo de grafos

El algoritmo que propone Guzmán et al. [31], se divide en dos algoritmos, el Algoritmo 3.10 muestra la construcción de conjuntos máximales independientes y el método general se muestra en el Algoritmo 2.6.

Algoritmo 2.6 Obtener_Coloreo(G)

Entrada: G una gráfica no dirigida

Salida: Un valor aproximado para $\chi(G)$

1: k = 0;
2: $I_B = \{u \in V(G) : u$ no es parte de cualquier ciclo impar de G
3: $G = G - I_B$
4: **while** is_bipartite(G)==false **do** {While there are odd cycles in G}
5: $T = CONSTRUIR\_MIS(G)$
6: $G = G - T$
7: $k = k + 1$ {Actualizar el siguiente MIS}
8: **end while**
9: $G = G \cup I_B$ { Regresar las componentes bipartitas}
10: 2-colouring(G){Al final, la gráfica restante es 2-coloreable}
11: **return** $\chi(G)$ es $k + 2$

El algoritmo 2.6 inicia una variable denominada $k = 0$ que contendrá al final el número de colores con el que es posible colorear los vértices de la gráfica G. Posteriormente, se eliminan caminos o árboles, denominado proceso de depuración de la gráfica que se realiza en las líneas dos y tres, del Algoritmo 2.7. Una vez depurada la gráfica, debido a que este tipo de gráficas usan dos colores, se construye el conjunto máximal independiente con el Algoritmo 2.8, los vértices contenidos en éste se eliminan de la gráfica y se aumenta una unidad a k. La construcción iterativa del MIS termina hasta que se obtiene una gráfica bipartita, por lo que al final k aumenta dos unidades.

Algoritmo 2.7 Depurar_Grafo(G)

Entrada: G una gráfica simple no dirigida

Salida: G con solo ciclos

1: sT = spanningTree(G)
2: cT = coTree(G, sT)
3: vCaminos=ObtenerCamino(cT, sT)
4: Eliminar de G los vértices pertenecientes a $vCaminos$
5: **return** G

El Algoritmo 2.7, se inicia construyendo un árbol de expansión de G, este se obtiene mediante el algoritmo de *Búsqueda en profundidad (Depht-first search)*

(ver Sección 2.2), posteriormente en la línea dos se obtiene un co-árbol, es decir, aquella gráfica resultante de las aristas que forman parte de ciclos básicos fundamentales en la gráfica original. Los vértices que no se encuentran en los ciclos básicos fundamentales, son los que se almacenan en $vCaminos$ los cuales se eliminan de la gráfica G, por lo que se regresa una gráfica G depurada.

En el Algoritmo 2.8 de la linea 1 a la 5 se obtienen vértices de no articulación (es decir, vértices que si se eliminan no desconectan la gráfica). Subsecuentemente entre las lineas 7 a la 12 se construye un MIS.

$N_{G_{j-1}}(x)$ es el conjunto de vecinos de cada vértice de no articulación. Si el resultado de la intersección entre el conjunto de vecinos de x y el actual MIS (K_0) es $\emptyset$, entonces x se agrega a K_0. De otra manera, en la linea 13 se obtienen los grados de cada vecino en orden descendiente, denominados como ne.

El siguiente paso es tomar cada elemento en ne, denominado $elem$, el cual forma un nuevo conjunto de vecinos, si la intersección entre este conjunto y el conjunto K_0 es $\emptyset$, entonces el $elem$ se agrega a K_0 y la iteración se rompe. El vértice con el grado mayor se considera para permitir cubrir la gráfica con un mínimo conjunto de colores.

Dado que agregar un vecino al MIS puede dividir la gráfica restante en dos partes (una condición no deseable en el algoritmo) se incluye un procedimiento de prueba. En el Algoritmo 2.8 en la línea 21 se lleva a cabo una prueba para comprobar si cada vértice en el conjunto K_0 no rompa la gráfica, en caso de que si la divida, se eliminan de K_0.

Algoritmo 2.8 CONSTRUIR_MIS(G)

Entrada: G una gráfica no dirigida

Salida: K_0 es un MIS tal que $\delta_G(K_0) \geq |V(G)| - 1$

```
1: i=0
2: while |E(G_i)| > |V(G_i)| do {Proceso de contracción}
3:     Seleccionar un vértice de no articulación x ∈ G_i
4:     Agregar x a la pila V y remover x de G_i
5:     G_{i+1} = G_i − {x}
6: end while
7: K_0 = ∅
8: repeat {Proceso de construcción de MIS}
9:     Obtener x de la pila V
10:    if N_{G_{j−1}}(x) ∩ K_0 = ∅ then
11:        K_0 = K_0 ∪ {x}
12:    else
13:        ne = Ordenar(N_{G_{i+1}}(x))
14:        for elem in ne do
15:            if N(elem) ∩ K_0 = ∅ then
16:                K_0 = K_0 ∪ {elem}
17:                break
18:            end if
19:        end for
20:    end if
21: until Pilas vacías
22: Verificar si algún vértice es de articulación, si un vértice rompe la gráfica removerlo de K_i.
23: return K_0 {Hasta este punto δ_G(K_0) ≥ |V(G)| − 1}
```

Capítulo 3

Marco Metodológico

La metodología que se siguió para propósito del desarrollo de la investigación consta de las etapas mostradas en la Figura 3.1.

En este capítulo se abordaran las etapas que corresponden a la mejora del algoritmo determinista explicado en la Sección 2.3.2.1, así como parte de la implementación de *E2COL* de la cual surge una modificación a dicho algoritmo, así como también una propuesta final de algoritmo para aproximar el coloreo, del diagrama de flujo mostrado en la Figura 3.1.

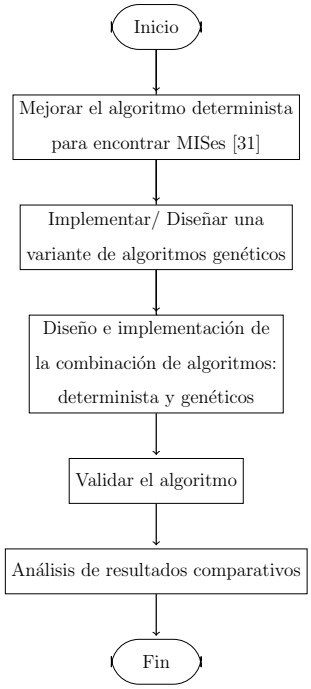


Figura 3.1: Diagrama de flujo de la metodología seguida

3.1. Mejora de un algoritmo determinista para encontrar MISes

De manera general el algoritmo determinista que se mejoró en esta Tesis, está dividido en las siguientes etapas:

- Depuración de la gráfica inicial.
- Construcción de conjuntos máximales independientes de la gráfica depurada.

Para precisar el algoritmo determinista, se divide en dos etapas, la primera se muestra en el Algoritmo 3.9 que es el proceso principal, mientras que el Algoritmo 3.10 presenta la construcción de los conjuntos máximales independientes.

Algoritmo 3.9 Coloreo_Grafo(G)

Entrada: G una gráfica simple no dirigida

Salida: Un valor aproximado para $\chi(G)$

1: k = 0;
2: $I_B = \{u \in V(G) : u$ no es parte de un ciclo impar de $G\}$
3: $G = G - I_B$
4: **while** is_bipartite(G)==false **do** {Mientras haya ciclos impares en G}
5: $T = Construir\_MIS(G)$
6: $G = G - T$
7: $k = k + 1$ {Actualizar el siguiente MIS}
8: **end while**
9: $G = G \cup I_B$ {Regresa la primera componente bipartita }
10: 2-Coloreo(G){Al final, el grafo restante es dos coloreable}
11: **return** $\chi(G)$ es $k + 2$

El Algoritmo 3.9 recibe una gráfica simple, es decir una gráfica sin aristas paralelas y no dirigido, la línea 1 del algoritmo inicializa la variable $k = 0$, la cual al final indicará el coloreo de la gráfica G. El proceso de depuración de la gráfica se realiza en las líneas 2 y 3, el cual es el mismo que se muestra en el Sección 2.3.2.1. De la línea 4 a la 8 se realiza la construcción y eliminación iterativa de los conjuntos máximales independientes de la gráfica G, hasta obtener una gráfica bipartita, en está parte se encuentra la mejora con respecto a [31]. Por cada iteración se aumenta una unidad la variable k, al final en la línea 9 los vértices de I_B se incorporan a la gráfica residual bipartita, para poder colorear

con dos colores la gráfica (línea 10), por lo que en la línea 11 el valor aproximado de $\chi(G)$ está dado por $k+2$ las dos unidades se aumentan ya que toda gráfica bipartita puede ser coloreada con dos colores [9].

Algoritmo 3.10 Construir_MIS(G)

Entrada: G una gráfica simple no dirigida y una matriz dinámica de no vecinos por vértice

Salida: K_0 es un MIS tal que $\delta_G(K_0) \geq |V(G)| - 1$

1: Seleccionar un vértice de no articulación $x \in G$
2: Agregar x al conjunto K_0
3: Obtener los no vecinos de x en G.
4: $noNeighbours$ = Ordenar los no vecinos de x de acuerdo al grado de manera ascendente.
5: **while** $noNeighbours$ **do**
6: $ve = noNeighbours[0]$
7: **if** ve es un vértice de no articulación **then**
8: Agregar ve al conjunto K_0
9: Borrar los vecinos de ve en $noNeighbours$.
10: Remover ve de $noNeighbours$
11: **end if**
12: **end while**
13: **return** K_0 {Hasta ahora $\delta_G(K_0) \geq |V(G)| - 1$}

El Algoritmo 3.10 realiza la construcción iterativa de los conjuntos máximales independientes, proceso mejorado con respecto al presentado en [31]. Este algoritmo recibe una gráfica simple no dirigida formada por ciclos, el primer paso consiste en seleccionar un vértice de no articulación x, siguiendo el proceso del Algoritmo 3.11, el cual será el primer vértice que forma parte del MIS. Posteriormente, se obtienen los no vecinos de x, esta lista contiene los candidatos a formar parte del MIS, se ordenan de manera ascendente de acuerdo al grado de cada vértice, se recorre esta lista y se verifica que el vértice ve no corte la gráfica para poder agregarlo al MIS, se obtienen los vecinos y éstos se eliminan de los candidatos. El algoritmo regresa un conjunto máximal independiente.

Algoritmo 3.11 Obtener_noArticulacion

Entrada: G una gráfica simple no dirigida

Salida: Vértice de no articulación

```
1: sT = spanningTree(G)
2: cT = coTree(G, sT)
3: vCaminos=ObtenerCamino(cT, sT)
4: vGrados=ObtenerGrados(cT)
5: while True do
6:     vNoArticulacion=Max(vCaminos, vGrados)
7:     if vNoArticulacion no corta la gráfica then
8:         return vNoArticulacion
9:     else
10:        Seleccionar el siguiente vértice más grande
11:    end if
12: end while
```

El Algoritmo 3.11 recibe una gráfica $G = (V, E)$ y regresa un vértice de no articulación, si es que existe. El proceso inicia generando un árbol de expansión de la gráfica G, así como su respectivo co-árbol. La manera en cómo se identifica un vértice de articulación es a través de identificar los vértices que más intervienen en algún ciclo de la gráfica, este proceso se realiza en la línea 3. Una vez creado el árbol de expansión y el co-árbol se genera un arreglo denominado $vCaminos$ de tamaño $|V|$ se inicia en ceros, donde cada posición del arreglo es un vértice de la gráfica que contiene el número de veces que un vértice participa en un ciclo. Se toma cada arista del co-árbol cT y se obtiene el camino en el árbol de expansión entre el par de vértices de la arista, cada vértice que forma parte del camino contabiliza en una unidad el valor almacenado en $vCaminos$, el arreglo que se construye en la línea 4 contiene el grado de cada vértice del co-árbol. En la línea 6 se obtiene el vértice que contenga el valor máximo en el arreglo $vCaminos$, si existe más de un vértice con el mismo valor, la selección del candidato será con el arreglo $vGrados$, seleccionando el de mayor grado entre los seleccionados de $vCaminos$.

3.2. Variante de $E2COL$

Derivado del proceso de investigación, se realizaron modificaciones al algoritmo propuesto por Wu. et al. [63], en la Fase de extracción, así como en la búsqueda Tabú y Fase de expansión. El Algoritmo 3.12 muestra como queda el algoritmo modificado.

De manera general, las modificaciones realizadas al Algoritmo $E2COL$ [63] son en la Fase de extracción, modifica la forma de obtener MISes; se modifica el método de búsqueda Tabú, para solucionar conflictos involucrados en un camino, así como el algoritmo propuesto solo considera la fase de extracción y la de coloreo. Al final de la fase de expansión, si no se encuentra un coloreo legal, se agregan clases vacías hasta lograr una solución con cero conflictos.

Algoritmo 3.12 Variante de $E2COL$

Entrada: Una gráfica no dirigido $G_0 = (V_0, E_0)$; un entero k

Salida: Un $k-$*coloreo* legal de G_0 o reporte de falla

1: {Fase de Extracción}
2: { En cada iteración de la extracción se eliminan conjuntos máximales independientes $I_0, \cdots, I_t$} $\{i = 0\}$
3: **while** $|V_i| > q$ **do**
4: G_{i+1}= Remover(G_i, I_i) {Remover todos los vértices de I_i de G_i}
5: $i = i + 1$
6: **end while**
7: { Fase de coloreo inicial}
8: s_i = Tabu_Search$(G_i, k - i)$ { Usar Tabu_Search para colorear la gráfica G_i con $k-1$ colores, s'_i = es el mejor coloreo encontrado en términos de la función f}
9: **if** s_i = es un $(k-i)$coloreo, es decir $f(s_i) = 0$ **then**
10: Usar s_i para construir un $k$coloreo $s_0 = \{s_i, I_{i-1}, \cdots, I_0\}$ de G_0
11: Regresar el s_0 y detener el proceso
12: **end if**
13: {Fase de Expansión }
14: **while** $i > 0$ **do**
15: $i = i - 1$
16: $s_i = \{s_{i+1}, I_i\}$ {Construir un $(k-i)$-coloreo de G_i de $s_{i+1} = \{C_1, \cdots, C_{k-i-1}\}$}
17: s_i =Tabu_Search(s_i, G_i)
18: **if** s_i es un $(k-i)$ coloreo, es decir $f(s_i) = 0$ **then**
19: Usar s_i para construir un k-coloreo legal $s_0 = \{s_i, I_{i-1}, \cdots, I_0\}$ de G_0
20: Regresar s_0 y detener el proceso
21: **end if**
22: **end while**
23: **while** $f(s_i) \neq 0$ **do**
24: $s_i = \bigcup\{\}$ {Se agrega una clase vacía a la solución}
25: s_i =Tabu_Search(s_i, G_i)
26: **end while**

La estrategia propuesta sigue el mismo principio que en [63] utiliza para la fase de extracción, dado un tamaño de MIS inicial, encontrar tantos MISes como sea posible para al final sólo quedarse con los MISes disjuntos, de tal manera

que, se asegura no tener vértices repetidos en MISes diferentes. El Algoritmo 3.13 describe la manera en como se obtiene un MIS de la misma cardinalidad.

Algoritmo 3.13 *obtencionMISES*

Entrada: Una gráfica no dirigida $G = (V, E)$

Salida: Un conjunto de MISes disjuntos de igual cardinalidad

```
1:  K = [ ]
2:  while |V| > q do
3:      for componente in G do
4:          if componente ! = aislado: then
5:              m ← extraerMIS(componente)
6:              Almacenar cada m obtenido por componente
7:          else
8:              Almacenar cada vértice aislado permitido en un conjunto
9:          end if
10:     end for
11:     Identificar la componente con el mayor número de MISes
12:     for componente ≠ componente con mayor número de MISes do
13:         Agregar los vértices de un MIS a un MIS del de mayor número de MISes
14:     end for
15:     Agregar aleatoriamente a un MIS los vértices aislados
16:     Eliminar de la gráfica los vértices de los MISes
17:     Agregar MISes a K
18: end while
19: return K
```

El Algoritmo 3.13 obtiene MISes hasta dejar una gráfica residual con q vértices, si una gráfica consta de diferentes componentes, se obtendrá por cada componente conectada un conjunto de MISes de una misma cardinalidad (línea 3-6). En la línea 10 se identifica qué componente tiene el mayor número de MISes obtenidos, para el resto de las componentes los MISes obtenidos se unirán con el conjunto de MISes más grande, en caso de ser un vértice aislado se almacenará en un conjunto para al final agregarlo aleatoriamente a un MIS, al final se almacenan los MISes obtenidos por cada iteración en K.

El Algoritmo 3.13 en la línea 4, hace uso del método *extraccionMIS*, descrito en el Algoritmo 3.14.

Algoritmo 3.14 *extraccionMIS*

Entrada: Una gráfica no dirigida $G = (V, E)$

Salida: Conjunto de MISes disjuntos

```
1: kmis ← encontrarMIS(graph)
2: candidatos ← [ ]
3: Agregar kmis a candidatos
4: for v in V do
5:     I_v = []
6:     Agregar v a I_v
7:     for w in N_G(v) do
8:         for x in N_G(w) do
9:             if (x,v) ∉ E then
10:                Verificar que x no comparta arista con algún otro vértice de I_v
11:                Agregar x a I_v
12:            end if
13:            if |I_v| == |kmis| then
14:                if I_v not in candidatos then
15:                    Agregar I_v a candidatos
16:                    break
17:                end if
18:            end if
19:        end for
20:    end for
21: end for
22: return obtenerDisjuntos(candidatos)
```

El Algoritmo 3.14 en un inicio busca un MIS, de acuerdo a la cardinalidad de $kmis$ se buscará generar conjuntos máximales independientes de la misma cardinalidad, cada conjunto será agregado a un conjunto denominado *candidatos*. De la línea 5 a la veintidós se encuentran tantos MISes de la cardinalidad de $kmis$ como sea posible, esto se logra generando un MIS denotado como I_v, tomando como inicio cada vértice v de la gráfica. Posteriormente, se obtienen los vecinos de v y por cada vecino se vuelven a obtener sus vecinos y éstos son candidatos a formar parte del MIS, siempre y cuando no se comparta arista con algún vértice del conjunto I_v. De la línea 14 a la 19 se verifica que I_v tenga el mismo tamaño que $kmis$, en caso de que lo tenga, I_v se agrega al conjunto de MISes candidatos y se intenta con otro vértice. Al finalizar en la línea 23 se regresan los MISes disjuntos de acuerdo a la función $obtenerDisjuntos$ (Algoritmo 3.17)

El Algoritmo 3.13 en la línea 1, implementa el Algoritmo descrito en 3.15 denominado *encontrarMIS*.

Algoritmo 3.15 *encontrarMIS*

Entrada: Una gráfica no dirigida $G = (V, E)$

Salida: Un MIS de mayor cardinalidad encontrado

```
1: mis ← crearMIS(G)
2: i = 0
3: while i < 60 do
4:     misDOS ← crearMIS(G)
5:     if |mis| < |misDOS| then
6:         mis ← misDOS
7:     end if
8:     i ← i + 1
9: end while
10: return mis
```

El Algoritmo 3.15 obtiene un MIS que en 60 iteraciones sea el de mayor cardinalidad, mediante la creación de MISes de manera voráz con el Algoritmo 3.16, verificando en cada iteración que mis sea en cuanto a cardinalidad más grande que $misDOS$, si $misDOS$ es de mayor cardinalidad, remplazará al mis actual, como se observa en las líneas 5 a 7. En la línea 1 y 4 se implementa el Algoritmo 3.16 para generar un MIS.

Algoritmo 3.16 *crearMIS*

Entrada: Una gráfica no dirigida $G = (V, E)$

Salida: Un conjunto máximal independiente

```
1: copiaVertices ← V
2: MIS ← []
3: while copiaVertices do
4:     v ← Random(copiaVertices)
5:     Agregar v a MIS
6:     Elimina v de copiaVertices
7:     vecinos ← N_G(v)
8:     for v' in vecinos do
9:         if v' in copiaVertices then
10:            Eliminar v' en copiaVertices
11:        end if
12:    end for
13: end while
14: return MIS
```

El Algoritmo 3.16 genera un MIS de manera vóraz, se realiza una copia de vértices en el conjunto *copiaVertices*, de la línea 3 a la 13, se genera el MIS, obte-

niendo un vértice v aleatoriamente del conjunto *copiaVertices* y se agrega a MIS, se elimina de *copiaVertices* y se obtienen los vecinos de v para ser eliminados de *copiaVertices* (línea de la 9 a la 10), este proceso se realiza hasta ya no tener más vértices en *copiaVertices*.

El Algoritmo 3.14 en la última línea llama a *obtenerDisjuntos*, el cual dado un conjunto de MISes candidatos, obtendrá todos los MISes disjuntos, asegurando que ningún vértice se repita en algún conjunto máximal independiente.

Algoritmo 3.17 *obtenerDisjuntos*

Entrada: Una gráfica no dirigida $G = (V, E)$, conjunto de MISes candidatos

Salida: MISes disjuntos

```
1: K = []
2: i = 0
3: while i < (|candidatos| − 1) do
4:     c1 ← candidatos[i]
5:     dc1 = []
6:     j = i + 1
7:     while j < |candidatos| do
8:         dck = []
9:         Agregar c1 a cdk
10:        k = j
11:        while k < |candidatos| do
12:            C2=candidatos[k]
13:            verificar intersección de c1 con c2
14:            verificar que c2 sea disjunto con los conjuntos dck
15:            if dck not in dc1 then
16:                Agregar dck a dc1
17:            end if
18:        end while
19:    end while
20:    Agregar dc1 a K
21: end while
22: return El conjunto de MISes más grande de K
```

El Algoritmo 3.17 en general obtiene por cada MIS candidato un conjunto de MISes disjuntos, es decir, dado el MIS $c1$, todos los conjuntos disjuntos con $c1$ que pertenecen a *candidatos* (línea 7 a 18). Al final se obtiene el conjunto de MISes más grande que se haya generado.

Una vez obtenida una gráfica residual de q vértices, para esta investigación el valor de q se obtiene $|V|-((|V|*60)/100)$, es decir, se considera un 60 % de vertices de la gráfica original que se deben colorear en la fase de Extracción mientras que

el 40 % de vértices quedan para la gráfica residual, para realizar el coloreo inicial, de las lineas 7 a 12, el coloreo se realiza con el Algoritmo de búsqueda Tabú, el cual se explica en el Algoritmo 3.18, el cual regresará una solución a la que se comprueba que tenga cero colisiones, para sólo agregar los MISes extraídos de la primera fase y regresarla como solución válida, caso contrario entra a la Fase de Extracción.

La búsqueda Tabú propuesta en esta investigación es la que se muestra en el Algoritmo 3.18 y es una variación de la búsqueda Tabú denominada *tabucol* propuesta por Hertz et al. [34], las variaciones son:

1. El vértice que se selecciona para mover de clase no se selecciona aleatoriamente, como originalmente se plantea, sino que en esta propuesta será un vértice que cause conflicto en la solución, es decir aquel vértice que comparte una arista con cualquier otro vértice en la misma clase.

2. La selección de la clase a la cual se moverá el vértice, en el algoritmo original se plantea que sea aleatorio entre todas las clases permitidas, sin embargo, se propone que sea a la clase con menos conflictos de las permitidas.

3. La búsqueda Tabú se itera hasta L veces dado por $|E| * 0.25$ (se considera el 25 % de aristas), intentando mover aleatoriamente vértices, sin embargo debido a que la propuesta opta por mover un vértice conflicto a una clase permitida con menos conflictos, se da el caso para vértices con todas las clases de la solución actual marcadas como tabú, esto indica que justo el vértice conflictivo colapsa y es necesario agregar una nueva clase y reiniciar la búsqueda tabú.

La búsqueda Tabú de manera general consta de los siguientes pasos:

1. Obtener los vértices conflicto de la solución s.

2. Se agrega a la lista tabu:

   - Vértice a mover identificado como $vMove$.
   - La clase de la que viene.
   - Tiempo de permanencia. (para este caso se propone $t = random(0, 9) + (0.6 * f(s))$, obtenido empíricamente por [24])

$$tabuList\{vertice : [[clase], [tiempo]]\}$$

3. Cada $vMove$ se mueve a una clase con menor número de conflictos y se elimina de la clase donde estaba.

4. Se evalúa $f(s)$, si es igual a cero se termina el proceso, caso contrario continua con el proceso.

Algoritmo 3.18 Algoritmo de búsqueda Tabú (*Tabu search*)

Entrada: Una gráfica no dirigido $G = (V, E)$; una solución s_0

Salida: El mejor coloreo s^* encontrado

1: $s^* = s_0$
2: $cont = 0$
3: $tabuList = \{\}$ {Inicializar la lista Tabú}
4: **while** $cont < L$ **do**
5: $nbCFL = f(s_0)$
6: Obtener los vértices conflicto de s_0
7: Obtener $vMove$, es decir un vértice a mover de clase
8: **if** $vMove$ en $tabuList$ **then**
9: Obtener clases permitidas, es decir, aquellas que no se encuentran en la lista tabú
10: **if** Hay clases permitidas **then**
11: Mover el vértice $vMove$ a la clase con menos conflictos permitida
12: $tl = \lceil Random(0, 9) + \alpha * nbCFL \rceil$
13: Agregar a $tabuList$ la clase en donde estaba $vMove$ por tl iteraciones
14: **else**
15: {Se decide si parar el proceso por un vértice que colapse}
16: Analizar vecinos de $vMove$ y decidir si continuar o no el proceso
17: **end if**
18: **else**
19: {$vMove$ no está en tabuList}
20: Mover el vértice $vMove$ a la clase con menos conflictos
21: $tl = \lceil Random(0, 9) + \alpha * nbCFL \rceil$
22: Agregar a $tabuList$ la clase en donde estaba $vMove$ por tl iteraciones
23: **end if**
24: Actualizar lista tabú, es decir reducir los tiempos tl
25: **if** $f(s_0) < f(s^*)$ **then**
26: $f(s^*) \leftarrow f(s_0)$
27: **end if**
28: **end while**
29: Regresar s^*

El Algoritmo 3.18 recibe una solución s_0 la cual al inicio se considera como la mejor solución (s^*), se inicializa un contador denominado *cont* de control que llegará hasta L iteraciones, donde $L = |E| * 0.25$, es decir se considera el 25 % de densidad de la gráfica para iterar la búsqueda Tabú. De la línea 4 a la 28 se itera L

veces la búsqueda tabú en cada iteración se realiza el proceso de mover un vértice conflicto, primero se evalúa la solución para obtener el número de conflictos, posteriormente se obtienen los vértices que causan conflicto, para obtener un vértice candidato a mover $vMove$ (línea 7) se realiza el siguiente proceso:

1. Se genera una lista considerando los vértice conflicto, por cada vértice conflicto se crea un arreglo con todos los vértices considerados conflicto con los que tiene una arista y que no han sido recorridos.

2. Se entra a un proceso recursivo, en el cual se obtiene un camino de vértices conflicto para el cual se trata de resolver en el camino los vértices intermedios.

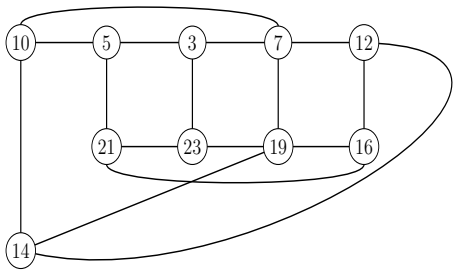


Figura 3.2: Gráfica de ejemplo.

Por ejemplo, si se tiene la solución s_0= [[3, 5, 7, 14, 16, 19, 21], [10, 12, 23]] de la gráfica de la Figura 3.2, los vértices conflicto de la solución son:

[3, 5, 7, 14, 16, 19, 21]

Por lo que la lista que se genera es:

3	[5, 7, 14, 16, 19, 21]
5	[16, 19, 21, 14, 7]
7	[16, 19, 21, 14]
14	[16, 19, 21]
16	[19, 21]
19	[21]
21	[ ]

El primer elemento que se considera para formar el camino de vértices en conflicto para este ejemplo es 3, de acuerdo a la lista, el vértice 3 comparte arista

con los vértices $[5, 7, 14, 16, 19, 21]$, el vértice 3 se almacena en una pila, para formar el camino, se toma el primer vértice de la lista anterior, el cual es 5 y se obtiene la lista de vértices $[16, 19, 21, 14, 7]$, el vértice 5 se almacena en la pila, el siguiente vértice a mover es el 16 con la lista de vértices $[19, 21]$, almacenando en la pila el vértice 16, ahora el vértice 19 con la lista $[21]$, por último el vértice 21 ya no tiene vértices que no hayan sido visitados anteriormente, por lo que se ingresa a la pila y el proceso regresa la pila de vértices.

Para obtener los vértices candidatos a mover de clase, es decir $vMove$ se saca de la pila el vértice 21, éste no se considera, el siguiente elemento a sacar es el 19 el cual se convierte en $vMove$ y se aplica el proceso de la línea 10 a la 23, el cual consiste en verificar si dicho vértice esta en la lista Tabú, en caso de que no este, es decir es nuevo en la lista Tabú (líneas 18 a 22) se obtiene la clase con menos conflicto, claro está que se podrá mover a cualquier otra clase que no sea en la que está actualmente, se calcula el tiempo que permanece en la lista Tabú y se crea el elemento. Por ejemplo 19 : $[[0], [9]]$ en la lista Tabú, es decir que para el vértice 19, se le prohibe regresar a la clase 0 en 9 iteraciones, en caso de que el vértice ya exista en la lista Tabú simplemente se obtienen las clases prohibidas y estas no se consideran como clase a mover y de nuevo se busca la clase a la que tenga menos conflictos el vértice (línea 10 a 13), en caso de que un vértice tenga todas las clases en la lista Tabú se analizan los vecinos, el Algoritmo 3.19 muestra como se toma la decisión de continuar o parar la búsqueda Tabú.

Algoritmo 3.19 Algoritmo que verifica vecinos

Entrada: Vértice que colapsa $vMove$, una gráfica no dirigida G, una solución s_0

Salida: Decide si continuar o no el proceso de la búsqueda Tabú

```
1: AA= N_G(vMove)
2: if Hay vecinos de vMove then
3:     coloresP = 2
4:     while AA do
5:         y = Obtener el primero elemento de conjunto AA
6:         AA = AA ⋂ N_G(y)
7:         if AA ≠ ∅ then
8:             coloresP = coloresP + 1
9:         end if
10:        if coloresP > |s_0| then
11:            return No continuar con Tabu_Search
12:        end if
13:    end while
14: end if
```

El Algoritmo 3.19 define si el proceso de búsqueda tabú continua, debido a

que un vértice colapsa con todas las clases que tiene la solución s_0, de manera general trata de encontrar vértices vecinos del vértice que colapsa $vMove$ que participen en una sub-gráfica completa. El proceso obtiene los vecinos del vértice que mantiene conflicto y se asignan al conjunto AA, es importante señalar que este conjunto está ordenado de manera ascendente (línea 1), mientras existan elementos en en conjunto AA (De la línea 4 a la 13) se obtiene el primer vértice denominado y, para el cual se realiza la intersección del conjunto AA con los vecinos de y dejando el resultado en el mismo conjunto AA (línea 6). Si el conjunto AA aún cuenta con elementos se aumenta una unidad la variable $coloresP$, la cual contabiliza el número de clases con las que se colorea la sub-gráfica completa, se inicia en 2 (línea 3) debido a que si existen elementos en el conjunto AA todos estos deben tener un color diferente a $vMove$, de la línea 10 a la 12 se verifica que la búsqueda Tabú continue o no, por lo que si $coloresP$ es mayor al número de clases de s_0 el proceso debe parar ya que $vMove$ seguirá en conflicto. De esta manera se busca que no exista un vecindario que cumpla con una sub-gráfica completa que no sea mayor al número de clases de la solución actual s_0, ya que por definición, una gráfica completa tiene como número cromático el número de vértices, por lo que si se obtiene una sub-gráfica completa con más vértices que número de clases en s_0, el proceso de búsqueda tabú debe terminar e intentar una nueva clase de color.

Una vez asignada una clase a $vMove$ continuando con el ejemplo, $vMove = 19$, en la pila el siguiente vértice no se considera, se vuelve a sacar un vértice y este nuevo se convierte en $vMove$. Para el ejemplo, el siguiente vértice es 5 al que se le tiene que asignar una nueva clase, el proceso continua hasta no tener elementos en la pila, se actualiza la lista Tabú (línea 24) y se evalúa la solución (líneas 25 a 27) dejando en s^* la mejor solución obtenida hasta el momento, si la solución actual s_0 contiene menos conflictos que s^*, entonces s_0 remplaza la solución s^*.

El proceso de perturbación ocupa un mayor tiempo de ejecución, por lo que para la propuesta se omite el proceso completo de perturbación.

Una modificación más, se encuentra en que una vez terminada la fase de expansión, si no se encuentra una solución con cero conflictos, el algoritmo original regresa un mensaje de error de que no se encontró un k-coloreo legal, sin embargo en la propuesta se decide agregar clases vacías hasta lograr una solución con cero conflictos (lineas 23 a 25).

3.3. Propuesta

La nueva propuesta, de manera general consta de los siguientes pasos:

1. Requiere un k número de colores, obtenido de `http://mat.gsia.cmu.edu/`

`COLOR04/`

2. Fase de extracción.

   - Elimina MISes de la misma cardinalidad.
   - Se obtiene una gráfica residual con hasta q vértices.

   $$q = |V| - (|V| * 0.65) \tag{3.1}$$

3. Fase de coloreo.

   - Genera una población de 20 individuos [63], para colorear con $k - MISes$ con la gráfica residual
   - Selección de dos individuos aleatoriamente $s1$, $s2$.
   - Generar h, aplicando cruza GPX (Algoritmo 2.5) a $s1$ y $s2$.
   - Aplica búsqueda Tabú al individuo h y se obtiene una solución $s3$.
   - Se reemplaza la solución $s1$ o $s2$ con mayor número de colisiones por la solución $s3$.

4. Fase de regeneración
   Si al finalizar la fase de coloreo en la población no se encuentra una solución con 0 colisiones.

   - Seleccionar un MIS del conjunto extraído en la fase de extracción m.
   - Aumentar los vértices que contienen m a la gráfica residual para reconstruir un MIS m'.
   - Eliminar m del conjunto de MISes extraídos de la primera fase.
   - Agregar m' al conjunto de MISes extraídos de la primera fase.
   - Se aplica la fase de coloreo inicial con la nueva gráfica residual.

   En caso de que hasta el momento no se logre obtener un coloreo sin colisiones, se agregarán clases de color, es decir aumentar en una unidad la variable k y se ejecuta la fase de coloreo hasta obtener una solución sin conflictos.

3.3.1. Fase de extracción

La Fase de Extracción contribuye a que el 65 % de vértices de una gráfica sean asignados a conjuntos bien formados (MISes), para que el resto de vértices se les asigne un color en la siguiente fase.

El proceso de extracción de MISes es el siguiente:

- De la gráfica se obtiene un primer MIS de manera aleatoria.
- Con base al tamaño del primer MIS obtenido se generan otros MISes del mismo o mayor tamaño denominados *MISes candidatos*.
- De los MISes candidatos se obtienen MISes disjuntos.

De manera general el Algoritmo 3.20 muestra el proceso de extracción de MISes que se realiza hasta obtener una gráfica con no más de q vértices.

Algoritmo 3.20 Extraccion()

Entrada: Una gráfica $G = (V, E)$, $q = |V| - (|V| * 0.65)$

Salida: Un Conjunto de MISes (K) y una gráfica G'

1: $K = \emptyset$
2: $G' = G$
3: **while** $|V| > q$ **do**
4: $\quad K \cup MISES(G')$
5: $\quad G' = G' \setminus MISES(G')$
6: **end while**
7: **return** K, G'

El Algoritmo 3.20 requiere una gráfica y un valor de q obtenido de acuerdo a la Ecuación 3.1, se inicializa un conjunto denominado K el cual, al final del proceso contendrá los MISes de la fase de extracción. Para obtenerlos, se copia la gráfica G a G' para eliminar vértices que participan en un MIS. El proceso iterativo de obtener MISes y eliminar los vértices contenidos en estos conjuntos, se realizan de la línea 3 a la 4. El proceso denominado $MISES$ se describe en el Algoritmo 3.21.

Algoritmo 3.21 MISES()

Entrada: Una gráfica $G = (V, E)$

Salida: Un conjunto de MISes M

1: $G' = G$
2: $C = \emptyset$
3: $v = random(V)$
4: $I = construyeI(G', v)$
5: Agregar I a C
6: **while** v in V **do**
7: $I' = construyeI(G', v)$
8: **if** $|I'| \geq |I|$ **then**
9: Agregar I' a C
10: **end if**
11: **end while**
12: **return** disjuntos(C)

El Algoritmo 3.21 requiere de una gráfica, el proceso copia la gráfica a G', se crea un conjunto C que almacenará los MISes candidatos a formar parte de la fase de extracción, se obtiene un vértice aleatorio v del conjunto de vértices V, el proceso $construyeI()$ genera un conjunto máximal independiente a partir de un vértice, este proceso se describe en el Algortimo 3.22, el primer conjunto I formado se almacena en C, posteriormente por cada vértice de la gráfica se construye un MIS (denominado I'), si la cardinalidad del conjunto I' es mayor o igual a la cardinalidad de I se almacena I' en C, al final el proceso $disjuntos()$ regresa únicamente los conjuntos maximales independientes disjuntos almacenados en C.

Algoritmo 3.22 ConstruyeI()

Entrada: Una gráfica $G = (V, E)$ y un vértice v

Salida: Un Conjunto Máximal Independiente (MIS)

1: $I = \emptyset$
2: **while** V **do**
3: $nn = (V \setminus N_G(v)) \setminus v$
4: V= $V \setminus N_G(v)$
5: Agregar v a I
6: **while** nn **do**
7: $w = random(nn)$
8: Agregar w a I
9: $nw = (V \setminus N_G(w)) \setminus w$
10: V= $V \setminus N_G(w)$
11: nn = $nn \setminus N_G(w)$
12: nn= $nn \cup nw$
13: **end while**
14: $v = random(V)$
15: **end while**
16: **return** I

El Algoritmo 3.22 describe el proceso de obtener un conjunto maximal independiente denominado I a partir de un vértice v, para el cual se obtiene los no vecinos (línea 3), se eliminan los vértices vecinos de v de la gráfica (línea 4) y se agrega el vértice v al conjunto I, una vez que se tienen los no vecinos de v se buscará agregarlos al conjunto I, tomando un no vecino aleatorio denominado w, se agrega al conjunto I y se eliminan de la gráfica los vecinos de w, así como también del conjunto nn, los no vecinos de w de agregan al conjunto nn, este proceso se realiza hasta completar la asignación de vértices pertenecientes al conjunto nn y continuar buscando un vértice aleatorio de la gráfica para agregar sus no vecinos al conjunto I hasta que los vértices de la gráfica hayan sido asignados.

La fase de extracción obtiene un conjunto de MISes y una gráfica sin los vértices participantes en los conjuntos maximales independientes denominada gráfica residual, la cual contendrá hasta q vértices, con la finalidad de que en la fase de coloreo inicial el proceso de coloreo inicial se facilite para una gráfica más pequeña que la inicial.

3.3.2. Fase de coloreo

La fase de coloreo se basa en un algoritmo genético, debido a que se considera una población, una representación de una posible solución, una función objetivo, pro-

ceso de selección, cruza y mutación, sin embargo, este proceso se modifica por una búsqueda aleatoria, se retoma el algoritmo propuesto por Hao and Galinier [24] (Algoritmo 2.3).

3.3.3. Fase de regeneración

Una vez que el proceso de coloreo haya terminado y no logró encontrar una solución con cero conflictos, se ingresa a la fase de regeneración, la cual toma el conjunto de MISes extraídos en la primera fase (Fase de extracción) y reconstruye un MIS, esta idea surge debido a que la construcción de MISes al inicio generó una gráfica residual para la cual el coloreo de $k-|MISes|$ no es el adecuado y requiere más clases de color, por lo cual se retoma un MIS, los vértices pertenecientes se agregan a la gráfica residual y se busca la construcción de un nuevo MIS. El Algoritmo 3.23 muestra el proceso completo de regenerar un MIS.

Algoritmo 3.23 RMIS()

Entrada: Una gráfica $G = (V, E)$, un entero k, M conjunto de MISes

Salida: La mejor solución encontrada

1: **while** no condicionParo() **do**
2: $I = random(M)$
3: $M \setminus I$
4: $nI = ConstruirMIS(G, I)$
5: $M \cup nI$
6: HEA(G, k)
7: **end while**

El Algoritmo 3.23 requiere el conjunto de MISes extraídos de la primera fase (M), una gráfica y el número de colores con los que se colorea la gráfica (k), se selecciona un MIS del conjunto de M (I) y se elimina del conjunto M (línea 3), esta selección es de manera aleatoria, el proceso de *construirMIS* se explica en el Algoritmo 3.24, el cual regresa un nuevo conjunto maximal independiente denominado nI, así como la gráfica sin aquellos vértices que forman parte del nuevo MIS, el conjunto nI formará parte del conjunto de MISes extraídos en la primera fase, por lo que se agrega al conjunto M (línea 5).

Por último se vuelve a ejecutar la fase de coloreo, es decir se implementa el algoritmo *HEA* (Algoritmo 2.3). Si en todo el proceso se encuentra una solución con cero conflictos se regresa esa solución, en caso contrario, si al terminar las iteraciones (100) aún no se encuentra una solución se aumenta el número de clases con $k = k + 1$ y se ejecuta la fase de coloreo, hasta encontrar una solución sin conflictos.

Algoritmo 3.24 construirMIS()

Entrada: Una gráfica $G = (V, E)$, un conjunto de vértices I

Salida: Un conjunto maximal independiente I'

1: $V = V \cup I$

2: $I' = generarMIS(G)$

3: $V = V \setminus I'$

4: **return** I'

El Algoritmo 3.24 genera un conjunto maximal independiente dada una gráfica y un conjunto de vértices, los cuales se agregaran a la gráfica (línea 1), posteriormente se implemente el algoritmo determinista propuesto por Guzmán et. al [32] descrito en la Sección 3.1, los vértices del conjunto maximal independiente denominado I' son eliminados de la gráfica, la cual se utilizará para aplicar el algoritmo HEA. El proceso finaliza regresando el conjunto I', el cual será parte del conjunto M.

Capítulo 4

Resultados experimentales

En este capítulo se presenta un análisis comparativo con respecto a los resultados obtenidos con JGraphT [49] (biblioteca de desarrollo para Java) y el algoritmo propuesto por Guzmán et al. [32] (Algoritmo de aproximación basado en la obtención de MISes. Apéndice A) y las propuestas de este trabajo de Tesis.

4.1. Gráficas utilizadas

El conjunto de gráficas en formato DIMACS empleadas en las prueblas fueron tomadas del repositorio (`http://mat.gsia.cmu.edu/COLOR04/` Tabla 4.1), las cuales se describen como:

- REG: Representa problemas basado en asignación de registros para variables en códigos reales, de Gary Lewandowski (gary@cs.wisc.edu).
- SGB: Son gráficas del conjunto de datos de Donald Knuth's en Stanford, éstas se dividen en:
  - *Book Graphs*: Basando en obras literarias, una gráfica se crea mediante la representación de cada vértice como un carácter. Dos vértices comparten arista si el correspondiente carácter se encuentran uno con el otro. Knuth creo gráficas para 5 obras literareas: Tolstoy's Anna Karenina (anna), Dicken's David Copperfield (david), Homer's Iliad (homer), Twain's Huckleberry Finn (huck) y Hugo's Les Misérables (jean).
  - *Queen Graphs*: Es una gráfica con mn vértices en la cual cada vértice representa un cuadrado en un tablero $m \times n$, dos vértices se conectan por una arista, si los correspondientes cuadrados están en la misma fila, columna o diagonal.

- CAR: Gráficas *k-insertion y full-insertion* son una generalización de gráficas *mycel* con vértices insertados para incrementar el tamaño de la gráfica pero no la densidad (es decir qué tan conectada se encuentra una gráfica).

De la mayoría de las gráficas consideradas para las pruebas, se tiene registrado el número cromático. Para el caso de esta Tesis se consideraron 24 instancias de las gráficas anteriormente descritas.

En la Tabla 4.1 se muestran las instancias con su correspondiente número de vértices y aristas originales. Considerando el Algoritmo 3.12 en la fase de extracción, se obtiene una gráfica residual de hasta q vértices, este valor se calcula como $|V| - ((|V| * 65)/100)$.

Tabla 4.1: Configuración de gráficas consideradas para realizar pruebas comparativas.

	Instancia	**Vértices**	**Aristas**	**q**
1	1-Insertions_4.col	67	232	23
2	2-Insertions_3.col	37	72	13
3	2-Insertions_4.col	149	541	52
4	3-Insertions_3.col	56	110	20
5	anna	138	986	48
6	david	87	812	30
7	fpsol2_i_1	269	11654	94
8	fpsol2_i_2	363	8691	127
9	fpsol2_i_3	363	8688	127
10	queen5_5	25	320	9
11	queen6_6.col	36	580	13
12	queen7_7.col	49	952	17
13	queen8_8.col	64	1456	22
14	queen9_9.col	81	2112	28
15	queen8_12	96	1368	34
16	queen11_11.col	121	3960	42
17	queen12_12.col	144	5192	50
18	queen13_13.col	169	6656	59
19	queen14_14.col	196	8372	69
20	queen15_15.col	225	10360	79
21	queen16_16.col	256	12640	90
22	zeroin_i_1	126	4100	44
23	zeroin_i_2	157	3541	55
24	zeroin_i_3	157	3540	55

En la siguiente sección se realiza un estudio comparativo de los resultados obtenidos para las 24 gráficas tomadas de muestra y ejecutadas en JGraphT, el algoritmo propuesto por Guzmán et al. [31] y las propuesta realizadas en esta tesis.

4.2. Comparativa de coloreo

Con las gráficas descritas en la sección anterior se realizaron pruebas comparativas con algoritmos que aproximan el coloreo de gráficas, entre éstos se encuentran el propuesto por Guzmán et al. [32], JGraphT [49] y la modificación de $E2COL$ descrito en el Algoritmo 3.12 de la Sección 3.2, así como los resultados obtenidos por la propuesta descrita en el Sección 3.3.

En la Tabla 4.2 se muestran los resultados obtenidos de aplicar a cada instancia los algoritmos de aproximación al número cromático, en la primera columna se indica el número de instancia, seguido del nombre de la instancia, en la tercer columna se muestra el valor exacto de coloreo, el resultado obtenido por JGraphT en la cuarta columna, mientras que en la quinta columna el resultado obtenido por Guzman et al. [32], el resultado obtenido por la propuesta de una variante de $E2COL$ se muestra en la sexta columna, mientras que el resultado obtenido por la propuesta final se muestra en la última columna.

Tabla 4.2: Coloreo de Gráficas

	Instancia	k	**JGraphT**	**Guzmán et. al. [32]**	**Propuesta Inicial**	**Propuesta**
1	1-Insertions_4.col	4	5	5	4.85 (0.366)	4 (0.000)
2	2-Insertions_3.col	4	4	4	4 (0.000)	4 (0.000)
3	2-Insertions_4.col	4	5	5	4.95 (0.224)	4 (0.000)
4	3-Insertions_3.col	4	5	4	4 (0.000)	4 (0.000)
5	anna	11	11	11	11.15 (0.366)	11 (0.000)
6	david	11	11	11	11.85 (0.489)	11 (0.000)
7	fpsol2_i_1	65	65	65	67.45 (1.317)	65 (0.000)
8	fpsol2_i_2	30	30	30	32.5 (0.761)	30 (0.000)
9	fpsol2_i_3	30	30	30	32.65 (1.040)	30 (0.000)
10	queen5_5	5	7	6	5 (0.000)	5 (0.000)
11	queen6_6.col	7	10	9	9 (0.324)	7.95 (0.224)
12	queen7_7.col	7	12	11	8.35 (1.309)	7.95 (0.759)
13	queen8_8.col	9	15	12	10.95 (0.394)	10.15 (0.366)
14	queen9_9.col	10	15	13	12 (0.324)	11.4 (0.598)
15	queen8_12	12	17	15	13.95 (0.224)	13.35 (0.489)
16	queen11_11.col	11	18	16	14.95 (0.394)	11.4 (0.598)
17	queen12_12.col	*	20	17	16.4 (0.598)	15.1 (0.641)
18	queen13_13.col	13	22	18	17.5 (0.513)	16.35 (0.587)
19	queen14_14.col	*	24	21	18.95 (0.224)	17.2 (0.696)
20	queen15_15.col	*	24	21	19.9 (0.447)	18.45 (0.686)
21	queen16_16.col	*	27	22	20.9 (0.308)	20.25 (0.851)
22	zeroin_i_1	49	49	49	49.45 (0.510)	49 (0.000)
23	zeroin_i_2	30	30	30	32.05 (0.945)	30 (0.000)
24	zeroin_i_3	30	30	30	31.85 (0.745)	30 (0.000)

De la Tabla 4.2 se pueden hacer estudios comparativos en pares, primero comparar los resultados obtenidos por JGraphT [49] con los obtenidos por Guzmán et al. [32]. De los resultados se puede observar lo siguiente: comparando JGraphT con el algoritmo de Guzmán et al. [32], en un 54.16 % de las instancias (es decir, en 13 instancias: 3-Insertions_3.col, queen5_5, queen6_6.col, queen7_7.col,queen8_8.col, queen9_9.col, queen8_12, queen11_11.col, queen12_12.col, queen13_13.col, queen14_14.col, queen15_15.col, queen16_16.col), este último mejora el resultado obtenido por JGraphT, mientras que en el 46 % restante de gráficas se igualan los resultados.

Por otra parte, comparando el algoritmo de Guzmán et al. [32] con el algoritmo híbrido, los resultados muestran que en un 41.66 % el algoritmo híbrido obtiene una mejor aproximación, la mayoría de los casos de éxito se encuentran en las gráficas denominadas *queen*. Mientras que en un 29.16 % los resultados son iguales en ambos algoritmos.

De acuerdo a la desviación estándar (escrita en paréntesis) obtenida en cada instancia de la primera propuesta híbrida, se muestra el coloreo promedio obtenido dentro de 20 ejecuciones, para el 62.5 % de las instancias, el coloreo son casi uniformes. Debido a que cada método implementado, tienen una metodología diferente de solución, la combinación de algoritmos genéticos y algoritmos deterministas asegura una aproximación al coloreo debido a que el 62.5 % de las instancias tienen una desviación estándar menor a 0.5, esto indica que la variabilidad de resultados es mínima.

De los resultados obtenidos en la última columna, con respecto a la primera propuesta híbrida (penúltima columna), podemos concluir que en un 54.16 % de las instancias consideradas en las pruebas los resultados obtenidos por la propuesta presentada en esta tesis son uniformes, debido a que los resultados en las 20 ejecuciones se obtiene el mismo coloreo, que es igual a la mejor aproximación registrada, debido a que la desviación estándar es igual a cero, esto indica que no hay variación en los resultados del coloreo en cada ejecución. Sin embargo en el 45.84 % de las gráficas la aproximación del coloreo es menor que la primera propuesta híbrida, por la desviación estándar mostrada, se indica que la variación de coloreo es mínima en cada ejecución.

Comparando entre todos los resultados obtenidos por cada algoritmo, en general se puede observar que la mejor aproximación de coloreo obtenida le corresponde a la propuesta de esta tesis (última columna), debido a que en un 54.16 % el coloreo obtenido es igual al registrado, mientras que en para el 45.84 % el coloreo es menor en en comparación al obtenido por JGraphT, Guzmán et al. y primera propuesta híbrida, sin embargo dicho resultado no queda tan alejado del mejor registrado, debido a que aumenta aproximadamente de una a cuatro unidades el coloreo.

4.2.1. Tiempo de ejecución

El análisis empírico de un algoritmo en términos de tiempo de ejecución, resulta ser interesante derivado de los diferentes algoritmos que aproximan el coloreo, debido a que cada uno de ellos varia en metodologías. Por ello, en la Tabla 4.3 se muestran los tiempos en segundos, que toma en obtener una aproximación al número cromático de una gráfica.

En general, en la Tabla 4.3 se puede observar que el algoritmo que obtiene

Tabla 4.3: Resultados de ejecución en segundos

	Instancia	**JGraphT**	**Guzmán et. al. [32]**	**Propuesta Inicial**	**Propuesta**
1	1-Insertions_4.col	0.1	0	1.8	14.3
2	2-Insertions_3.col	0.1	0	0	0.0
3	2-Insertions_4.col	0.3	0.1	20.7	29.6
4	3-Insertions_3.col	0.1	0	0	0.0
5	anna	0.1	0.2	1.2	0.5
6	david	0.2	0.1	2.7	36.7
7	fpsol2_i_1	1.2	28.3	69.4	1183.1
8	fpsol2_i_2	0.9	11	311	383.7
9	fpsol2_i_3	1.2	11.1	185.9	477.0
10	queen5_5	0.1	0	0.1	16720.1
11	queen6_6.col	0.1	0	2.6	2391.9
12	queen7_7.col	0.1	0.1	0.6	4719.8
13	queen8_8.col	0.2	0.2	12.2	4495.6
14	queen9_9.col	0.3	0.3	22.9	6506.2
15	queen8_12	0.5	0.4	12.5	8814.9
16	queen11_11.col	0.3	0.9	58.6	12896.2
17	queen12_12.col	0.3	1.3	138	14017.4
18	queen13_13.col	0.4	2.2	160.2	10226.0
19	queen14_14.col	0.6	3.6	306.3	8633.7
20	queen15_15.col	0.6	5.3	448.9	16954.1
21	queen16_16.col	0.7	7.7	633.3	12488.2
22	zeroin_i_1	1.1	3.6	3.8	2.9
23	zeroin_i_2	0.7	2.1	12.2	4.3
24	zeroin_i_3	0.9	2.1	13.5	1012.4

resultados de manera más rápida es JGraphT, sin embargo en términos generales éste no obtiene la mejor aproximación al coloreo de una gráfica respecto a las demás propuestas. Por otra parte, comparando el tiempo de ejecución tanto el algoritmo propuesto por Guzmán et al. y el algoritmo híbrido para instancias con $|V| < 100$ y $|E| < 1000$, el tiempo que toman en obtener una solución es similar, pero para instancias con $|V| > 100$ y $|E| > 1000$, el algoritmo híbrido

toma más del doble de tiempo que utiliza Guzmán et al. y en algunos casos obtienen los mismo resultados. Por ejemplo en la gráfica fpsol2_i_2, ambos obtienen aproximadamente 30 colores.

El tiempo de ejecución obtenido por la propuesta final de esta tesis para el 83 % de instancias, el tiempo obtenido es mayor que el obtenido por el resto de algoritmos analizados, sin embargo la aproximación al coloreo es mejor.

Capítulo 5

Conclusiones y Trabajo a futuro

En este capítulo se presentan las conclusiones que se obtuvieron al final de la investigación derivado de los resultados experimentales de la Sección 4, así como también los trabajos a futuro a partir del resultado final de la presente Tesis.

5.1. Conclusiones

El problema del coloreo de gráficas por su importancia se ha tratado de solucionar por diferentes algoritmos, entre los más usados se encuentran los heurísticos, los cuales aproximan la solución, además existen diferentes metodologías de solución. En particular, el presente trabajo de Tesis se ha centrado en tres algoritmos heurísticos, el algoritmo *greedy* de JGraphT, el algoritmo determinista propuesto por Guzmán et al. y las propuestas híbridas de combinar un algoritmo determinista con un algoritmo genético.

Derivado de los resultados obtenidos y presentados en la Sección 4, se puede concluir que en términos de número de colores las propuestas de Guzmán et al.(Apéndice A) y la que se presenta como trabajo final de tesis son competitivas, debido a que, el promedio de colores obtenidos en 20 ejecuciones, los resultados obtenidos por la propuesta híbrida de esta tesis, son mejores en gran medida que los obtenidos por Guzmán et al. debido a que el número de colores no está disperso con respecto al promedio obtenido, esto se observa en la desviación estándar obtenida para cada prueba. Adicionalmente, de los resultados de coloreo obtenidos por JGraphT el 62.5 % excede el coloreo de una a diez unidades de color, mientras que la propuesta de esta Tesis en sólo 45.83 % de las instancias el coloreo excede de una a cinco unidades de color, mientras que para Guzmán et al. en un 58.33 % el coloreo obtenido excede de una a seis unidades de color.

Los resultados obtenidos en esta investigación y concluidos como los que man-

tienen una mejor aproximación al coloreo, se deben a la implementación de la Fase de regeneración, la cual favoreció en la mayoría de las instancias, que de la gráfica conocida como residual, a que la aproximación al coloreo se cumpliera con el número de colores restantes, es decir el número k conocido como el mejor coloreo obtenido hasta el momento, se le resta el número de MISes extraídos de la primera fase, debido a que los MISes de la fase de extracción son considerados de la gráfica completa, dejando una gráfica residual densa, la cual complica que el coloreo sea propio con $k - MISes$.

Por otro lado, de los análisis realizados en la Sección 4 se encuentra el de tiempo de ejecución, donde se muestra el tiempo en segundos que tardó cada algoritmo en obtener una aproximación, de esto se puede concluir que en términos de tiempo de ejecución el algoritmo híbrido propuesto, toma un mayor tiempo de ejecución en comparación con el resto de los algoritmos comparados, debido a que en la fase de regeneración se considera un número de iteraciones para reconstruir un MIS, así como también el volver a implementar la fase completa de coloreo, dentro de la cual el tiempo que se toma en ejecutar la búsqueda Tabú incrementa el tiempo de ejecución. Sin embargo, ya que el objetivo general fue mejorar la aproximación al coloreo, se considera que se cumplió el objetivo y se verificó la hipótesis propuesta.

5.2. Trabajo a Futuro

Este trabajo es un análisis preliminar de algoritmos heurísticos, cada uno con una metodología diferente, dejando una línea abierta de análisis en la búsqueda de una combinación de parámetros libres en la búsqueda Tabú, tales como la manera de asignar vértices conflicto a una nueva clase, así como el tiempo en el que permanece penalizado un vértice, por ejemplo se puede considerar la densidad de la gráfica para poder determinar el tiempo de penalización. Se sugiere también el considerar otras estrategias para escapar de óptimo locales, por ejemplo, considerar algún método de selección en el que se considere la función objetivo.

Debido a que el tiempo de ejecución es el más alto de todos los algoritmo comparados dadas las iteraciones que toma la búsqueda Tabú, el encontrar una solución para optimizar el tiempo es importante, en la búsqueda Tabú independientemente de la gráfica se tiene siempre un número de iteraciones, ante lo cual como linea abierta de investigación se sugiere considerar el tipo de gráfica con características tales como la densidad, el grado de los vértices, entre otro factores que ayuden a determinar el continuar arreglando conflictos o el parar debido a que el número de colores es insuficiente.

Por otra parte, cambiar el proceso de actualización de la población contribuiría

a tener un balance de posibles soluciones, de este modo se mantendría la calidad y diversidad en la población, por ejemplo Yin and Hao [35] proponen un función de evaluación para actualizar la población considerando el número de colisiones y la distancia entre soluciones, es decir qué tan parecidas son. Para la propuesta que se tiene hasta el momento, únicamente se reemplaza la peor solución con la solución generada.

Por último, se sugiere la implementación de otros algoritmos heurísticos como *Hill Climbing, Simulated annealing, colonia de hormigas*, entre otros, que también son usados para aproximar soluciones en gráficas.

Bibliografía

[1] Sage. Obtenido de: http://www.sagemath.org/. Consultada el 17 de Septiembre del 2015s.

[2] Ahmed, S. Applications of graph coloring in modern computer science. *International Journal of Computer and Information Technology (IJCIT) 3*, 2 (2013), 1–7.

[3] Almara'beh, H., and Suleiman, A. Heuristic algorithm for graph coloring base on maximum independent set. *Journal of Applied Computer Science & Mathematics 6*, 13 (2012), 19–23.

[4] Beigel, R., and Eppstein, D. 3-coloring in time $o(1.3289^n)$. *Journal of Algorithms 54*, 2 (2005), 168–204.

[5] Björklund, A., and Husfeldt, T. Exact algorithms for exact satisfiability and number of perfect matchings. *Algorithmica 52*, 2 (2008), 226–249.

[6] Blöchliger, I., and Zufferey, N. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers & Operations Research 35*, 3 (2008), 960–975.

[7] Blum, A. New approximation algorithms for graph coloring. *Journal of the ACM (JACM) 41*, 3 (1994), 470–516.

[8] Bodlaender, H. L., and Kratsch, D. An exact algorithm for graph coloring with polynomial memory. Tech. rep., Technical report, Department of Information and Computing Sciences, Utrecht University, 2006.

[9] Bondy, J. A., and Murty, U. S. R. *Graph theory with applications*, vol. 290. Macmillan London, 1976.

[10] Brélaz, D. New methods to color the vertices of a graph. *Communications of the ACM 22*, 4 (1979), 251–256.

[11] Brownlee, J. Clever algorithms. *Nature-Inspired Programming Recipes* (2011), 436.

[12] Burke, E. K., Meisels, A., Petrovic, S., and Qu, R. A graph-based hyper heuristic for timetabling problems. *Computer Science Technical Report No. NOTTCS-TR-2004-9, University of Nottingham* (2004).

[13] Byskov, J. M. Chromatic number in time $o(2.4023^n)$ using maximal independent sets. *BRICS Report Series 9*, 45 (2002).

[14] Christofides, N. An algorithm for the chromatic number of a graph. *The Computer Journal 14*, 1 (1971), 38–39.

[15] Cormen, T. H. *Introduction to algorithms*. MIT press, 2009.

[16] De Ita Luna, G., and Castillo, J. A. Recognizing 3-colorings cycle-patterns on graphs. *Pattern Recognition Letters 34*, 4 (2013), 433–438.

[17] de Ita Luna, G., Marcial-Romero, J. R., and Moyao, Y. An approximate algorithm for the chromatic number of graphs. *Electronic Notes in Discrete Mathematics 46* (2014), 89 – 96. Jornadas de Matemática Discreta y Algorítmica.

[18] de Werra, D., Eisenbeis, C., Lelait, S., and Marmol, B. On a graph-theoretical model for cyclic register allocation. *Discrete Applied Mathematics 93*, 2–3 (1999), 191 – 203.

[19] Deo, N. *Graph theory with applications to engineering and computer science*. PHI Learning Pvt. Ltd., 2004.

[20] Eiben, A., van der Hauw, J., and van Hemert, J. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics 4*, 1 (1998), 25–46.

[21] Epp, S. *Discrete mathematics with applications*. Cengage Learning, 2010.

[22] Fleurent, C., and Ferland, J. A. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research 63*, 3 (1996), 437–461.

[23] Formanowicz, P., and Tanaś, K. A survey of graph coloring-its types, methods and applications. *Foundations of Computing and Decision Sciences 37*, 3 (2012), 223–238.

[24] Galinier, P., and Hao, J.-K. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization 3*, 4 (1999), 379–397.

[25] Gamache, M., Hertz, A., and Ouellet, J. O. A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding. *Computers & operations research 34*, 8 (2007), 2384–2395.

[26] Garey, M. R., Johnson, D. S., and So, H. C. An application of graph coloring to printed circuit testing. *Circuits and Systems, IEEE Transactions on 23*, 10 (1976), 591–599.

[27] Glover, F. W., and Kochenberger, G. A. *Handbook of metaheuristics*, vol. 57. Springer Science & Business Media, 2006.

[28] Goldberg, D. E., et al. *Genetic algorithms in search optimization and machine learning*, vol. 412. Addison-wesley Reading Menlo Park, 1989.

[29] Golumbic, M. C. *Algorithmic graph theory and perfect graphs*, vol. 57. Elsevier, 2004.

[30] Gross, J. L., and Yellen, J. *Graph theory and its applications*. CRC press, 2005.

[31] Guzmán-Ponce, A., Marcial-Romero, J. R., Ita, G. D., and Hernández, J. A. An algorithm to approximate the chromatic number of graphs. In *Electronics, Communications and Computers (CONIELECOMP) IEEE, 2015 International Conference on* (Feb 2015), pp. 110–115.

[32] Guzmán-Ponce, A., Marcial-Romero, J. R., Ita, G. D., and Hernández, J. A. Approximate the chromatic number of a graph using maximal independent sets. In *2016 International Conference on Electronics, Communications and Computers (CONIELECOMP) IEEE* (Feb 2016), pp. 19–24.

[33] Hao, J.-K., and Wu, Q. Improving the extraction and expansion method for large graph coloring. *Discrete Applied Mathematics 160*, 16–17 (2012), 2397 – 2407.

[34] Hertz, A., and de Werra, D. Using tabu search techniques for graph coloring. *Computing 39*, 4 (1987), 345–351.

[35] Jin, Y., Hao, J.-K., and Hamiez, J.-P. A memetic algorithm for the minimum sum coloring problem. *Computers & Operations Research 43* (2014), 318 – 327.

[36] Johnson, D. S., Aragon, C. R., McGeoch, L. A., and Schevon, C. Optimization by simulated annealing: an experimental evaluation; part ii,

graph coloring and number partitioning. *Operations research 39*, 3 (1991), 378–406.

[37] Karger, D., Motwani, R., and Sudan, M. Approximate graph coloring by semidefinite programming. *Journal of the ACM (JACM) 45*, 2 (1998), 246–265.

[38] Karp, R. M. *Reducibility among combinatorial problems*. Springer, 1972.

[39] Kempe, A. B. On the geographical problem of the four colours. *American journal of mathematics 2*, 3 (1879), 193–200.

[40] Knuth, D. E. *The art of computer programming: sorting and searching*, vol. 3. Pearson Education, 1998.

[41] Kocay, W., and Kreher, D. L. *Graphs, algorithms, and optimization*. CRC Press, 2004.

[42] Leighton, F. T. A graph coloring algorithm for large scheduling problems. *Journal of research of the national bureau of standards 84*, 6 (1979), 489–506.

[43] Lewis, R. R. *A Guide to Graph Colouring: Algorithms and Applications*, 1st ed. Springer Publishing Company, Incorporated, 2015.

[44] Lü, Z., and Hao, J.-K. A memetic algorithm for graph coloring. *European Journal of Operational Research 203*, 1 (2010), 241–250.

[45] Malaguti, E., and Toth, P. A survey on vertex coloring problems. *International Transactions in Operational Research 17*, 1 (2010), 1–34.

[46] Man, K.-F., Tang, K. S., and Kwong, S. *Genetic algorithms: Concepts and designs*. Springer Science & Business Media, 2012.

[47] Melanie, M. An introduction to genetic algorithms. *Cambridge, Massachusetts London, England, Fifth printing 3* (1999), 62–75.

[48] Moalic, L., and Gondran, A. *The New Memetic Algorithm $$HEAD$$ for Graph Coloring: An Easy Way for Managing Diversity*. Springer International Publishing, Cham, 2015, pp. 173–183.

[49] Naveh, B., and Contributors. Jgrapht. Obtenido de: http://jgrapht.org/. Consultada el 21 de Octubre de 2015.

[50] Omari, H. A., and Sabri, K. E. New graph coloring algorithms. *Journal of Mathematics and Statistics 2*, 4 (2007), 439.

[51] Plumettaz, M., Schindl, D., and Zufferey, N. Ant local search and its efficient adaptation to graph colouring. *Journal of the Operational Research Society 61*, 5 (2010), 819–826.

[52] Rahman, M. S. *Basic Graph Theory*, 1 ed. 1863-7310. Springer International Publishing, 2017.

[53] Ringel, G., and Youngs, J. W. Solution of the heawood map-coloring problem. *Proceedings of the National Academy of Sciences 60*, 2 (1968), 438–445.

[54] Sedgewick, R., and Wayne, K. *Algorithms*. Addison-Wesley Professional, 2011.

[55] Sivanandam, S., and Deepa, S. *Introduction to genetic algorithms*. Springer Science & Business Media, 2007.

[56] Smith, D., Hurley, S., and Thiel, S. Improving heuristics for the frequency assignment problem. *European Journal of Operational Research 107*, 1 (1998), 76–86.

[57] Trudeau, R. J. *Introduction to graph theory*. Courier Corporation, 2013.

[58] Vlasie, R. D. Systematic generation of very hard cases for graph 3-colorability. In *Proceedings of the Seventh International Conference on Tools with Artificial Intelligence* (Washington, DC, USA, 1995), TAI '95, IEEE Computer Society, pp. 114–119.

[59] Voloshin, V. I. Graph coloring: History, results and open problems.

[60] Welsh, D. J., and Powell, M. B. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal 10*, 1 (1967), 85–86.

[61] Wilson, R. A. *Graphs, colourings and the four-colour theorem*. Oxford University Press, 2002.

[62] Wilson, R. J. *An introduction to graph theory*. Pearson Education India, 1970.

[63] Wu, Q., and Hao, J.-K. Coloring large graphs based on independent set extraction. *Computers & Operations Research 39*, 2 (2012), 283–290.

[64] Xie, X.-F., and Liu, J. Graph coloring by multiagent fusion search. *Journal of combinatorial optimization 18*, 2 (2009), 99–123.

Apéndices

Apéndice A

Publicación

Approximate the chromatic number of a graph using maximal independent sets

Angélica Guzmán-Ponce*, J. Raymundo Marcial-Romero*, Guillermo De Ita †, José A. Hernández*
*Facultad de Ingeniería, Universidad Autónoma del Estado de México
aguzmanp643@alumno.uaemex.mx
jrmarcialr@uaemex.mx
xoseahernandez@uaemex.mx
†Facultad de Ciencias de la Computación, Benemérita Universidad Autónoma de Puebla
deita@solarium.cs.buap.mx

***Abstract*—In this paper, we present an algorithm to approximate the chromatic number of a graph. The proposed approach initially removes even cycles and acyclic subgraphs, this process is called *debugging*, later to colour the remaining graph, we present a strategy to obtain maximal independent sets. We experimentally show that our proposal improves the results compare to JGraphT and Sage which contain a module for computing the chromatic number of a graph.**

I. Introduction

Let $G = (V, E)$ be a graph with vertex set V and set of edges E, the graph colouring problem consists in assigning a colour to each vertex in V with the characteristic that if two vertices are adjacent (i.e. the vertices share an edge) they can not have the same colour. Graph colouring aims at finding the smallest k for a giving graph called *the chromatic number* $\chi(G)$. It is well none that $\chi(G) \leq \Delta(G) + 1$, where $\Delta(G)$ is the maximum degree of G.

The graph colouring problem is a NP-complete problem [**?**], [**?**], so reductions to other combinatorial problem can be described, for example the problem of schedule assignment, where the goal is to perform a schedule using the smallest number of possible hours, can be addressed by finding the minimum number of colours that the schedule will have [1].

Additionally, the problem of classroom and subjects assignment is similar to the schedule assignment, in this problem the smallest number of classrooms needed to cover the subject schedule is searched. Another example is related to mobile antennas, where each city has a set of antennas and each antenna attend a particular zone, however the antennas work with frequencies, so that each city represents a vertex and the antennas's frequency is represented by colours [2]. Other application could be computer register allocation [3], printed circuit board testing [4] and Satellite range scheduling [5].

To solve the graph colouring problem there are two trends, the first implies the use of exacts algorithms, in this case the algorithms developed have been characterized by their exponential complexity either in time or space with respect to the number of vertices or edges in the graph. For example Beigel et.al. [6] designed an algorithm to check whether a graph with n vertices is 3-colouring (i.e. the graph can be colouring with three colours). He established an $O(1.3289^n)$ time algorithm, independently the number of edges in the graph.

Bodlaender et.al. [7] designed an exact algorithm whose complexity in both time and in use of memory is $O(5.283^n)$.

Given the exponential complexity of the exacts algorithms, mainly to the problem of 3-colouring or above, there have been developed heuristic methods, which although not guarantee the exact solution, they tend to come close. For example De Ita et. al. [8] developed an algorithm based in patterns, where a basic cycles of a graph determines the 3-colouring graphs properties, in which the patterns are encoded by conjunctive forms (F_G). In this way they determine the search of 3-colouring graphs properties. The number of clauses and size of F_G depends on the number of basic cycles in the graph, so the algorithm determines whether a graph is 3-colouring according to the number of basic cycles given.

On the other hand Almara'beh H. et.al. [9] propose three heuristic algorithms to find the chromatic number of a graph based on the maximal independent set MIS. The first algorithm implies a selection of a vertex with minimum and maximum degree consecutively (Min_Max). The algorithm is implemented, tested and compared with two algorithms one which selects a vertex with minimum degree first ($SNMD$) and the other selects a vertex with maximum degree ($SNXD$). The obtained results showed that the Min_Max algorithm and ($SNXD$) are better than ($SNMD$), the result were based on the processing time and the approximation of the chromatic number obtained.

De Ita et. al. [10] propose an approximate algorithm for the chromatic number of graphs. The first step removes bipartite components I_B since they can be colouring at the end of the process with two colours $G_i = (G - I_B)$. The second step consists on building and removing maximal independent sets (MIS) from G_i. To do this, firstly a no articulation vertex $x \in G_i$ is chosen and pushed to a stack V of vertex and their incidents edges an other stack, remove x from G_i, this process is performed until $|E(G_i)| < |V(G_i)|$. The next step consist on iteratively bundling a MIS until a 3-colouring subgraph is reached.

In [11] an approach based on the principle of *"reduce-and-solve"* is presented. The strategy consist in applying prior to

978-1-5090-0079-1/16/$31.00 ©2016 IEEE

the phase of graph colouring a preprocessing procedure to extract maximal independent sets from a graph until the graph becomes "sufficiently" small and the residual graph can be "easy" to colouring with any algorithm.

In this paper, we present an improvement to the construction of maximal independent set (MIS) of the algorithm propose in [10]. Basically, our proposal consists in building a MIS from no articulation vertices of the highest degree which are not adjacent to it. Finally we show experimentally that the results obtained improve to well established implementations.

The rest of this paper is organized as follow. In Section II, we review definitions used in the article. In Section III we detailed the proposed algorithm to approximate the chromatic number of a graph. Section IV we presented results obtained by testing the proposal against JGraphT, Sage and the algorithm in [10]. Finally we presented a conclusion in section V.

II. Preliminaries

Let $G = (V, E)$ be an undirected simple graph (i.e. finite, loop-less and without parallel edges) with vertex set V and set of edges E. $E(G)$ and $V(G)$ emphasize the set of edges and vertices of a particular graph G respectively.

Two vertices v and w are called *adjacent* if there is an edge $\{v, w\} \in E$, joining them. The *neighbourhood* of $x \in V$ is $N(x) = \{y \in V : \{x, y\} \in E\}$ and its *closed neighbourhood* is $N(x) \cup \{x\}$ which is denoted by $N[x]$. Note that v is not in $N(v)$, but is in $N[v]$.

The cardinality of a set A, is denoted by $|A|$. Given a graph $G = (V, E)$, the degree of a vertex $x \in V$, denoted by $\delta(x)$, is $|N(x)|$. If A is a set of vertices from a graph G, $N(A)$ is the set of neighbour vertices from any vertex of A, that is, $N(A) = \cup_{x \in A} N(x)$, while $N[A] = N(A) \cup A$.

The maximum degree of G or just the degree of G is $\Delta(G) = max\{\delta(x) : x \in V\}$, while we denote with $\delta_{min}(G) = min\{\delta(x) : x \in V\}$ and with $\delta(G) = (2 \cdot |E|)/|V|$ the minimum and average degree of the graph respectively.

Given a graph $G = (V, E)$, $S \subseteq V$ is an independent set in G if for whatever two vertices v_1, v_2 in S, $\{v_1, v_2\} \notin E$. Let $I(G)$ be the set of all independent sets of G. An independent set $S \in I(G)$ is *maximal*, abbreviated as MIS, if it is not a subset of any larger independent set and, it is *maximum* if it has the largest size among all independent sets in $I(G)$. The *independence number* $\alpha(G)$ is the cardinality of the maximum independent set of G.

Given a subset of vertices $S \subseteq V(G)$ the subgraph of G denoted by $G|S$ has vertex set S and a set of edges $E(G|S) = \{\{u, v\} \in E : u, v \in S\}$. $G|S$ is called the *subgraph of G induced by S*. We write $G - S$ to denote the graph $G|(V - S)$. The subgraph induced by $N(v)$ is denoted as $H(v) = G|N(v)$ which has to $N(v)$ as the set of vertices and all edges upon them.

Given a subgraph $H \subseteq G$, for each vertex $x \in V(H)$, let $\delta_H(x)$ be the degree of x in the induced subgraph H of G, if $H = G$ then $\delta_G(x) = \delta(x)$ and $E_H(x) = \{\{x, u\} \in E(G) : u \in H\}$. Similarly, $N_H(x)$ denotes the set of vertices from H adjacent to x. For any subgraph $H \subseteq G$, $\delta_G(H) = \sum_{x \in H} \delta_G(x)$. If H is an independent set of G then $\delta_G(H)$ is the number of edges of G incident to any vertex of H.

A path from a vertex v to a vertex w in a graph is a sequence of edges: $v_0v_1, v_1v_2, \ldots, v_{n-1}v_n$ such that $v = v_0$, $v_n = w$,v_k is adjacent to v_{k+1} and the length of the path is n. A simple path is a path such that $v_0, v_1, \ldots, v_{n-1}, v_n$ are all distinct. A cycle is just a nonempty path such that the first and last vertices are identical, and a simple cycle is a cycle in which no vertex is repeated, except the first and last vertices.

A k-cycle is a cycle of length k, that is, a k-cycle has k edges. A cycle of odd length is called an odd cycle, while a cycle of even length is called an even cycle. A graph G is acyclic if it has not cycles.

A connected component of G is a maximal induced subgraph of G, that is, a connected subgraph which is not a proper subgraph of any other connected subgraph of G. Note that, in a connected component, for every pair of its vertices x, y, there is a path from x to y. If an acyclic graph is also connected, then it is called a tree.

Let G be a connected graph, a vertex $v \in V(G)$ is called a no articulation point if $G \setminus v$ is a connected graph. A subset $S \subset V(G)$ is called a no articulation set if $G \setminus S$ is a connected graph.

A colouring of a graph $G = (V, E)$ is an assignment of colours to its vertices. A colouring is *proper* if adjacent vertices always have different colours. A k-colouring of G is a mapping from V into the set $\{1, 2, \ldots, k\}$ of k "colours". The chromatic number of G denoted by $\chi(G)$ is the minimum value k such that G has a proper k-colouring.

If $\chi(G) = k$, G is then said to be k-chromatic. The value $\chi(G)$ is polynomial computable when $\chi(G) \leq 2$, but when $\chi(G) \geq 3$, the problem becomes NP-complete, even for graphs G with degree $\Delta(G) \geq 3$.

Let $G = (V, E)$ be a graph, G is a *bipartite graph* if V can be partitioned into two subsets U_1 and U_2, called *partite sets*, such that every edge of G joins a vertex of U_1 and a vertex of U_2.

If G is a k-chromatic graph, then it is possible to partition V into k independent sets $V_1, V_2, ..., V_k$, called *colour classes*, but it is not possible to partition V into $k-1$ independent sets.

III. Chromatic Number Procedure

We firstly present and explain the algorithm proposed in [10] since a modification of the algorithm is reported in this work. The algorithm is divided into two sub-algorithms to show the MIS construction (Algorithm 2) and the general method (Algorithm 1).

We firstly explain Algorithm 1. Line one initializes $k = 0$ since it is assumed that the graph is not two colourable otherwise there are polynomial time algorithms to do so. Additionally, a depth first search of the graph is built. At lines two and three, vertices in G which are not in odd cycles are discarded, thereby building a sub-graph $G_0 \subseteq G$. Naming to this process *debugging graph*. So that G_0 will be colouring at the end of the process with two colours. In line five the

Algorithm 1 Seek_Chromatic_Number(G)

Require: G a non directed graph
Ensure: An approximate value for $\chi(G)$
1: k = 0; G=dfs(G)
2: $I_B = \{u \in V(G) : u$ is not part of any odd cycle of G
3: $G = G - I_B$
4: **while** is_bipartite(G)==false **do** {While there are odd cycles in G}
5: $T = Build\_MIS(G)$
6: $G = G - T$
7: $k = k + 1$\{Updating for the next MIS\}
8: **end while**
9: $G = G \cup I_B$\{returns the first bipartite component \}
10: Call 2-colouring(G)\{At the end, the remaining graph is 2-colourable\}
11: **return** $\chi(G)$ is $k + 2$

algorithm calls to $BUILD\_MIS(G)$ where in each iteration it forms a MIS.

In each iteration of the algorithm 1 between lines four to eight, a MIS is defined as K_0 which is discarded of actual graph G_i and a new sub-graph $G_{i+1} = (G_i - K_i)$ is constructed, the process continues until a sub-graph can be 2-colourable. At the end of the process an approximate chromatic number of the graph is obtained, setting an average number $[\frac{\delta(G)}{2}] + 2$ colours [10] where $\delta(G)$ is the average degree of G.

In Algorithm 2 between lines one to five, the vertices named as *no articulation vertex* (i.e. vertex which if removed do not disconnect the graph) are chosen. Subsequently between lines seven to twelve a MIS is built.

$N_{G_{j-1}}(x)$ is the set of neighbours of each $no\ articulation\ vertex$. If the result of the intersection between the set of the neighbours of x and the actual MIS (K_0) is $\emptyset$ then x is added K_0 (between lines nine to eleven). Otherwise, in line thirteen a method which returns the sorted degrees of each neighbour in descending order, defined as ne, is called.

The next step is to take each element in ne, defined as $elem$, forming a new set of its neighbours, if the intersection between this set and the set K_0 is $\emptyset$, $elem$ is added into K_0 and the iteration is broken. The vertex with the highest degree is taken since it would allow to cover the graph with a minimum set of colours.

Since adding a neighbour to the MIS could lead to split the remaining graph into two parts (a no desirable condition in the original algorithm) a testing procedure was included. In algorithm 2 the line twenty one carry out a test to check whether each vertex in the set K_0 do not break the graph. Those vertices which breaks the graph, are removed form K_0.

The proposal in this paper is presented in Algorithm 3, which is the MIS construction. Algorithm 1 is the same for this proposal, except the procedure of the line five which builds a MIS.

Initially a no articulation vertex x is obtained, in line 2, x is stored in K_0. In lines 3 and 4, the no neighbour vertices of

Algorithm 2 BUILD_MIS(G)

Require: G a non directed graph
Ensure: K_0 is a MIS such that $\delta_G(K_0) \geq |V(G)| - 1$
1: i=0
2: **while** $|E(G_i)| > |V(G_i)|$ **do** {Contraction process}
3: choose a no articulation vertex $x \in G_i$
4: push x to a stack V and remove x from G_i
5: $G_{i+1} = G_i - \{x\}$
6: **end while**
7: $K_0 = \emptyset$
8: **repeat** {Building a MIS process}
9: pop x from stack V
10: **if** $N_{G_{j-1}}(x) \cap K_0 = \emptyset$ **then**
11: $K_0 = K_0 \cup \{x\}$
12: **else**
13: $ne = arrange(N_{G_{i+1}}(x))$
14: **for** elem in ne **do**
15: **if** $N(elem) \cap K_0 = \emptyset$ **then**
16: $K_0 = K_0 \cup \{elem\}$
17: break
18: **end if**
19: **end for**
20: **end if**
21: **until** stacks are empty
22: Check if some vertices are an articulation vertex, if a vertex break the graph remove from K_i.
23: **return** K_0 {At this point $\delta_G(K_0) \geq |V(G)| - 1$}

Algorithm 3 NEW_BUILD_MIS(G)

Require: G a non directed graph and a dynamic matrix
Ensure: K_0 is a MIS such that $\delta_G(K_0) \geq |V(G)| - 1$
1: Choose a no articulation vertex $x \in G$
2: Add x to a set K_0
3: Get no neighbours of x in G.
4: $noNeighbours$ = Sort no neighbours of x.
5: **while** $noNeighbours$ **do**
6: ve = noNeighbours[0]
7: **if** ve is a no articulation vertex **then**
8: Add ve to a set K_0
9: Delete neighbours of ve in noNeighbours.
10: Remove ve from noNeighbours
11: **end if**
12: **end while**
13: **return** K_0 {At this point $\delta_G(K_0) \geq |V(G)| - 1$}

x are sorted and stored in a set called $noNeighbours$.

Between lines 5 to 12 a candidate vertex ve to form part of K_0 is obtained, in line 6, the first element in $noNeighbours$ is chosen as the candidate, in line 7, it is verified whether ve breaks the graph, if this vertex does not break G, it is added to K_0 (line 8), subsequently, in lines 9 the neighbours of ve which belong to the set $noNeighbours$ are removed from the graph, since they cannot be part of the MIS. In line 10, ve is removed from $noNeighbours$ since it is already in

the MIS. The process between lines 5 to 12 is repeated until $noNeighbours$ is empty.

Compared with the algorithm at [10], the improvement consists on working with the $noNeighbours$ instead of the hole remaining set of vertices. As shown in the next section this strategy allows to improve the MIS construction and to improves the chromatic number.

A. Example

In this section, we present as example of our proposal the colouring of graph *game6* (Figure 1). According to Algorithm 1 the first step consists in debugging the graph, however for this example it is not necessary and initialize $k = 0$. The next step builds a MIS while the graph is not bipartite, *game6* is not bipartite, so the Algorithm 3 builds a MIS.

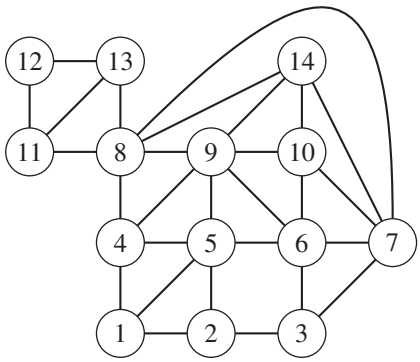


Fig. 1. Example Graph-*game6*

According to 3 the first step chooses a no articulation vertex, a method to obtain it, is by building a spanning tree and its co-tree, later on looking for a vertex with the greatest number of vertices in its path on the tree-cotree. In this case, a spanning tree Figure 2 and their co-tree Figure 3

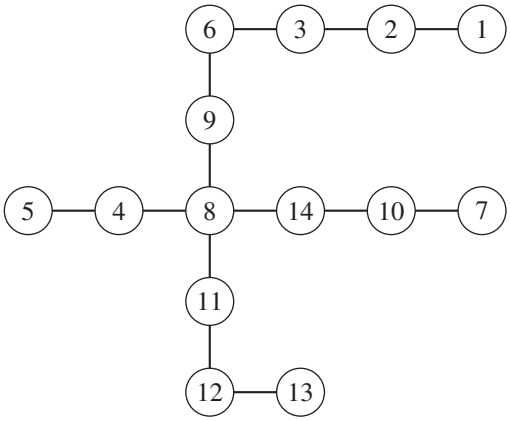


Fig. 2. Spanning Tree of *game6*

Considering the spanning tree and co-tree, a possible no articulation vertex is 8, but this vertex break the graph, so the next candidate is 9, and this vertex does not break the graph.

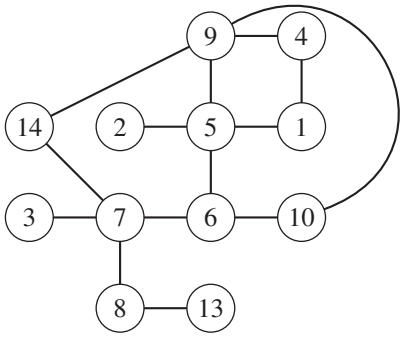


Fig. 3. Co-Tree of *game6*

As vertex 9 does not break the graph, this is added to K_0 and its no neighbours are obtained from the graph *game6*. The vertices are then sorted.

$noNeighbours$ = [7, 1, 2, 3, 11, 13, 12]

While $noNeighbours$ is not empty, choose the first vertex in $noNeighbours$, for this example, $ve = 7$, if the ve does not break the graph this vertex is added to K_0, as $ve = 7$ is a no articulation vertex, it is added, so $K_0 = \{9, 7\}$. The no neighbours of 7 are:

$neighbours$ = [3, 6, 8, 10, 14]

Next, the neighbours from noNeighbours and ve.

$noNeighbours$ = [1, 2, 11, 13, 12]

Iteratively obtain vertex while $noNeighbours$ is not empty, next vertex is $ve = 1$, this not break the graph so $K_0 = \{9, 7, 1\}$ and their neighbours are:

$neighbours$ = [2, 4, 5]

Next, the neighbours from noNeighbours and ve are deleted.

$noNeighbours$ = [11, 13, 12]

This process finishes when there are not vertices in $noNeighbours$, so in this example the first MIS is $K_0 = \{1, 7, 9, 11\}$. Each vertex in K_0 is removed from G (*game6*) and increase one unit k. In Figure 4 has a graph without vertices of K_0, as this graph is bipartite, stop the process and increase k in two units, because the remaining is two colourable.

At the end the chromatic number is 3.

IV. Testing

To compare this proposal, we decided to take the same graphs tested in [10] i.e graphs where the chromatic number is known. The first source is a game called Flood Fill [12], the base of the game is the theorem of four colours shown by Appel K et.al. [13], the theorem states that four colours are enough to color any map, analogously the game has maps, where each map has four colours maximum to colouring the

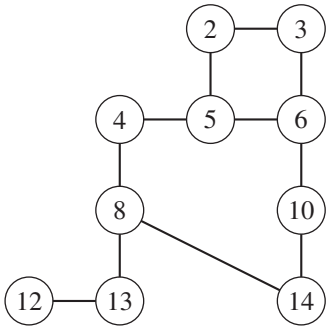


Fig. 4. Graph *game6* without vertices from K_0

graph, some maps could be colouring with less colours. The second source is a benchmark [14], which has graphs by Leighton [15]. The files with the graphs are identified by starting with the letters in allusion to the author of the graphs followed by a number which is 450, which is the number of vertices, followed by an underscore and a number, this is the exact chromatic number that can be used to colouring the graph, this graphs can be coloured with five, fifteen or twenty colours, on depending the version, the last letter of the file name varies between a,b,c or d, as this will increase the number of edges.

In [10] the graphs of the two sources where the input of software that approximate the chromatic number of a graph, in this case JGraphT and Sage. JGraphT [16] is a free software, available for Java and uses a greedy algorithm to obtain chromatic number. Sage [17] is a mathematical open source software which offers a module related to graph colouring and use linear programming .

In Table I the first column is *File name* which represents the graph's name, the first seventeen graphs reported are part of the game Flood Fill [12] and the last twelve graphs are part of the benchmark [14]. The second column has the results of [10] of testing the 28 cases, the result of testing graphs with JGraphT [16] and Sage is shown in third and four columns respectively .

The example named as *game12* represents a forest, which consists of two sub-graphs, the values shown at tables I, II which are separated by ";" represent the vertices, edges and chromatic numbers of each subgraph. It is important to said that neither JGraphT [16] nor Sage can accept a forest, so the result is not displayed.

In the game scenario, the algorithm in [10] is as competitive as the JGraphT and Sage implementations, specially equivalent to JGraphT. With respect to the benchmark, JGraphT approximate better to the exact solution. The testing could not compare results with Sage because the software do not get result for these graphs. In other words in 22 of them the proposed algorithm in [10] improves the results, i.e. 75% of them which means that the strategy improves the results in most of the cases.

To prove the improvement the Table I we present results with our proposal in the last column.

TABLE I
RESULTS OF COMPARATIVE TESTING

File name	Result of [10]	JGraphT	Sage	Our proposal
game1	2	2	2	2
game2	3	3	3	3
game3	3	3	3	3
game4	4	4	4	4
game5	3	3	3	3
game6	4	4	3	3
game7	3	4	3	4
game8	4	4	4	4
game9	4	4	4	4
game10	4	4	4	5
game11	3	4	3	3
game12	3,4	*	*	3;3
game13	4	3	3	3
game14	5	5	4	4
game15	4	4	3	4
game16	4	4	3	4
game17	4	3	3	3
le450_5a	12	12	*	10
le450_5b	13	12	*	10
le450_5c	16	12	*	10
le450_5d	14	13	*	10
le450_15a	22	18	*	18
le450_15b	21	18	*	17
le450_15c	31	26	*	24
le450_15d	32	26	*	25
le450_25a	28	26	*	26
le450_25b	29	26	*	26
le450_25c	37	30	*	29
le450_25d	37	30	*	29

The first seventeen graphs (Flood Fill's graphs) we report an improvement in five graphs that are *game6, game12, game13, game14 and game17* against the algorithm proposed in [10], also there is an improvement on the results in graphs *game6, game12 and game14* against JGraphT; with the rest of graphs we offer an equivalent chromatic number like Sage and JGraphT.

The testing with the benchmark [14] had better results against [10], and improvement the results of JGraphT in all the cases.

In Table II we report the time in seconds that the implementations [10], JGraphT, Sage and our proposal took to approximate the chromatic number of the graph. As noted, the time obtained by our proposal has been reduced compared to [10]. However for graphs with a bigger number of vertices an edges JGraphT has a better computing time, because JGraphT attempts to colour each vertex with a single colour each iteration, and these vertices will be removed from the graph at the end of each iteration.

TABLE II
RESULTS OF COMPARATIVE TESTING TIME IN SECONDS

File name	Result of [10]	JGraphT	Sage	Our proposal
game1	0.096349001	0.039	0.0007	0.000803232
game2	0.076768875	0.023	0.0004	0.001207113
game3	0.093842983	0.025	0.0004	0.069028854
game4	0.111377001	0.041	0.0007	0.076827049
game5	0.122199059	0.044	0.0007	0.074726105
game6	0.174742937	0.035	0.0006	0.003085852
game7	0.201543808	0.03	0.0005	0.009516954
game8	0.13003397	0.028	0.0005	0.068962097
game9	0.134248972	0.035	0.0006	0.003665209
game10	0.171635866	0.045	0.0008	0.005539894
game11	0.358479977	0.047	0.0008	0.010907888
game12	0.166887999	0.025	0.0004	0.071156979
game13	0.17260313	0.05	0.0008	0.018538952
game14	0.266232014	0.06	0.0010	0.076984167
game15	0.321135998	0.055	0.0009	0.010751009
game16	0.285405874	0.099	0.0017	0.007559061
game17	0.257588148	0.059	0.0010	0.006510019
le450_5a	139.618669	0.794	*	104.8809519
le450_5b	140.7595251	0.796	*	105.095856
le450_5c	370.208981	0.991	*	271.2560201
le450_5d	347.7004268	1.003	*	285.0784411
le450_15a	380.543427	0.79	*	240.008512
le450_15b	378.242377	0.891	*	237.7374761
le450_15c	2816.822651	1.36	*	1121.198253
le450_15d	2860.134034	1.938	*	1153.149893
le450_25a	479.1032259	0.878	*	265.7032979
le450_25b	487.0262668	0.867	*	269.426955
le450_25c	4274.227279	1.388	*	1269.447843
le450_25d	4836.956859	1.356	*	1296.873709

V. CONCLUSIONS

Compared with the solution at [10], JGraphT and Sage our proposal presents an improvement to approximate the chromatic number of a graph. The results are improved due to MIS construction strategy. Thus ensuring that there will be as many independent vertices in the set as in the set of no neighbors.

With the improvement, the aproximation of chromatic number obtained by our proposal gives better results in graphs with a bigger number of vertices an edges compared with JGraphT and Sage, and it is as competitive as JGraphT and Sage in graphs with a small number of vertices and edges.

VI. FUTURE WORK

A time-efficient way to obtain maximal independet sets, to equal or improve JGraphT in time, as well as improve the results to approximate the chromatic number of a graph with other strategy.

REFERENCES

[1] M. Gamache, A. Hertz, and J. O. Ouellet, "A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding," *Computers & operations research*, vol. 34, no. 8, pp. 2384–2395, 2007.

[2] S. Ahmed, "Applications of graph coloring in modern computer science," *International Journal of Computer and Information Technology (IJCIT)*, vol. 3, pp. 1–7, 2012.

[3] D. de Werra, C. Eisenbeis, S. Lelait, and B. Marmol, "On a graph-theoretical model for cyclic register allocation," *Discrete Applied Mathematics*, vol. 93, no. 23, pp. 191 – 203, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166218X99001055

[4] M. R. Garey, D. S. Johnson, and H. C. So, "An application of graph coloring to printed circuit testing," *Circuits and Systems, IEEE Transactions on*, vol. 23, no. 10, pp. 591–599, 1976.

[5] X.-F. Xie and J. Liu, "Graph coloring by multiagent fusion search," *Journal of combinatorial optimization*, vol. 18, no. 2, pp. 99–123, 2009.

[6] R. Beigel and D. Eppstein, "3-coloring in time $o(1.3289^n)$," *Journal of Algorithms*, vol. 54, no. 2, pp. 168–204, 2005.

[7] H. L. Bodlaender and D. Kratsch, "An exact algorithm for graph coloring with polynomial memory," Technical report, Department of Information and Computing Sciences, Utrecht University, Tech. Rep., 2006.

[8] G. De Ita Luna and J. A. Castillo, "Recognizing 3-colorings cycle-patterns on graphs," *Pattern Recognition Letters*, vol. 34, no. 4, pp. 433–438, 2013.

[9] H. Almara'beh and A. Suleiman, "Heuristic algorithm for graph coloring base on maximum independent set," *Journal of Applied Computer Science & Mathematics*, vol. 6, no. 13, pp. 19–23, 2012.

[10] *An algorithm to approximate the chromatic number of graphs*, Feb 2015. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7071022

[11] Q. Wu and J.-K. Hao, "Coloring large graphs based on independent set extraction," *Computers & Operations Research*, vol. 39, no. 2, pp. 283–290, 2012.

[12] "Microsiervos flood fill: El teorema de los cuatro colores, llevado a la prctica," Obtenido de: http://www.microsiervos.com/archivo/juegos-y-diversion/flood-fill-teorema-cuatro-colores.html, consultada el 04 de Agosto del 2014.

[13] K. I. Appel and W. Haken, *Every planar map is four colorable*. American mathematical society Providence, RI, 1989, vol. 98.

[14] C. J., "Graph coloring instances," Obtenido de: http://mat.gsia.cmu.edu/COLOR/instances.html#XXREG, consultada el 9 de Agosto de 2014.

[15] F. T. Leighton, "A graph coloring algorithm for large scheduling problems," *Journal of research of the national bureau of standards*, vol. 84, no. 6, pp. 489–506, 1979.

[16] B. Naveh and Contributors, "Jgrapht," Obtenido de: http://jgrapht.org/, consultada el 21 de Julio de 2014.

[17] "Sage," Obtenido de: http://www.sagemath.org/, consultada el 17 de Julio del 2014.